Volume Seven

XView Reference Manual

By Dan Heller

Updated for XView Version 3.2 by Thomas Van Raalte

O'Reilly & Associates, Inc.
**Preface**

<table>
<thead>
<tr>
<th>How to Use This Manual</th>
<th>xi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Font Conventions Used in This Manual</td>
<td>xii</td>
</tr>
<tr>
<td>Related Documents</td>
<td>xiii</td>
</tr>
<tr>
<td>Requests for Comments</td>
<td>xiv</td>
</tr>
<tr>
<td>Acknowledgements</td>
<td>xiv</td>
</tr>
</tbody>
</table>

**Chapter 1 XView Package Summary**

<table>
<thead>
<tr>
<th>1.1 XView and OPEN LOOK</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 XView Packages</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Object Handles</td>
<td>5</td>
</tr>
<tr>
<td>1.2.2 Attribute-based Functions</td>
<td>6</td>
</tr>
<tr>
<td>1.2.3 Creating and Manipulating Objects</td>
<td>6</td>
</tr>
<tr>
<td>1.3 The CANVAS Package</td>
<td>7</td>
</tr>
<tr>
<td>1.4 The CMS Package</td>
<td>8</td>
</tr>
<tr>
<td>1.5 The CURSOR Package</td>
<td>8</td>
</tr>
<tr>
<td>1.6 The DRAGDROP and DROP_SITE_ITEM Packages</td>
<td>9</td>
</tr>
<tr>
<td>1.7 The FONT Package</td>
<td>9</td>
</tr>
<tr>
<td>1.8 The FRAME Package</td>
<td>10</td>
</tr>
<tr>
<td>1.9 The FULLSCREEN Package</td>
<td>11</td>
</tr>
<tr>
<td>1.10 The ICON Package</td>
<td>12</td>
</tr>
<tr>
<td>1.11 The MENU and MENUTEM Packages</td>
<td>12</td>
</tr>
<tr>
<td>1.11.1 Menu Types</td>
<td>13</td>
</tr>
<tr>
<td>1.11.2 Menu Items</td>
<td>13</td>
</tr>
<tr>
<td>1.12 The NOTICE Package</td>
<td>14</td>
</tr>
<tr>
<td>1.13 The PANEL Package</td>
<td>15</td>
</tr>
<tr>
<td>1.13.1 Panel Items</td>
<td>15</td>
</tr>
<tr>
<td>1.13.2 Scrollable Panels</td>
<td>23</td>
</tr>
<tr>
<td>1.14 The SCREEN Object</td>
<td>23</td>
</tr>
</tbody>
</table>
### Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>XView class hierarchy</td>
<td>5</td>
</tr>
<tr>
<td>1-2</td>
<td>Panel item created with PANEL_BUTTON package</td>
<td>16</td>
</tr>
<tr>
<td>1-3</td>
<td>Panel button with an attached menu</td>
<td>16</td>
</tr>
<tr>
<td>1-4</td>
<td>Panel item created with the PANEL_ABBREV_MENU BUTTON package</td>
<td>17</td>
</tr>
<tr>
<td>1-5</td>
<td>Panel item from the PANEL_CHOICE package</td>
<td>17</td>
</tr>
<tr>
<td>1-6</td>
<td>Panel item from the PANEL_CHECK_BOX package</td>
<td>17</td>
</tr>
<tr>
<td>1-7</td>
<td>Panel item created with PANEL_TOGGLE macro</td>
<td>18</td>
</tr>
<tr>
<td>1-8</td>
<td>Panel item created with PANEL_CHOICE_STACK macro</td>
<td>18</td>
</tr>
<tr>
<td>1-9</td>
<td>Panel drop target items</td>
<td>19</td>
</tr>
<tr>
<td>1-10</td>
<td>Panel item created with the PANEL_GAUGE package</td>
<td>19</td>
</tr>
<tr>
<td>1-11</td>
<td>Panel list item created with the PANEL_LIST package</td>
<td>20</td>
</tr>
<tr>
<td>1-12</td>
<td>Panel item created with PANEL_MESSAGE package</td>
<td>20</td>
</tr>
<tr>
<td>1-13</td>
<td>Another panel item created with PANEL_MESSAGE package</td>
<td>21</td>
</tr>
<tr>
<td>1-14</td>
<td>Panel item created with the PANEL_MULTILINE_TEXT package</td>
<td>21</td>
</tr>
<tr>
<td>1-15</td>
<td>Panel item created with the PANEL_NUMERIC_TEXT package</td>
<td>22</td>
</tr>
<tr>
<td>1-16</td>
<td>Panel item created with the PANEL_SLIDER package</td>
<td>22</td>
</tr>
<tr>
<td>1-17</td>
<td>Panel item created with the PANEL_TEXT package</td>
<td>23</td>
</tr>
</tbody>
</table>
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1 Generic Functions</td>
<td>6</td>
</tr>
<tr>
<td>1-2 XView Packages</td>
<td>6</td>
</tr>
<tr>
<td>5-1 Event Codes</td>
<td>231</td>
</tr>
<tr>
<td>5-2 Mouseless Event Codes</td>
<td>234</td>
</tr>
</tbody>
</table>
This page intentionally left blank

to preserve original page counts.
This page intentionally left blank

to preserve original page counts.
This manual describes the XView attributes, callbacks, procedures, macros, data structures, as well as additional reference material. XView (X Window-System-based Visual/Integrated Environment for Workstations) is a user-interface toolkit to support interactive, graphics-based applications running under the X Window System. This toolkit, developed by Sun Microsystems, Inc., is derived from earlier toolkits for the SunView windowing system. With over 2000 SunView applications in the workstation market, there are many programmers already familiar with SunView application programmer’s interface (API).

XView is based upon Xlib, the lowest level of the X Window System available to the programmer. While developing XView user interfaces does not require Xlib programming experience, there are good reasons for learning more about Xlib, especially if your application renders graphics.

How to Use This Manual

The XView Reference Manual includes reference material for each of the XView attributes and procedures, information about XView resources, and data structures. Within each section, reference pages are organized alphabetically.

This book is designed to be used with the XView Programming Manual. The XView Programming Manual provides an explanation of XView, including tutorial material and numerous programming examples. To get the most out of the XView Programming Manual, you will need the exact arguments for the attributes and the calling sequence for the callbacks and procedures that this reference manual provides.

The following paragraphs briefly describe the contents of this book:

Section 1, XView Package Summary, provides an introduction to XView and provides a summary of each of the XView packages.

Section 2, XView Attributes, provides alphabetically arranged descriptions of all attributes, and the callback procedures that are supplied by the application programmer.

Section 3, Procedures and Macros, provides alphabetically arranged descriptions of all the XView procedures and macros.
Section 4, *Data Types*, lists the data types defined by XView.

Section 5, *Event Codes*, lists the event codes in numerical order by value.

Section 6, *Command-line Arguments and XView Resources*, lists the XView options that can be set using command-line options. This section also lists the resources that XView uses to define certain default values when an application is initialized.

Appendix A, *Selection Compatibility Attributes*, provides the attributes for the old selection mechanism. A new selection package has been added in XView Version 3. The new selection package supports the standard XView API. The old selection mechanism is still supported in XView Version 3.

Appendix B, *Selection Compatibility Procedures and Macros*, provides the procedures and macros for the old selection mechanism. A new selection package has been added in XView Version 3. The new selection package supports the standard XView API. The old selection mechanism is still supported in XView Version 3.

Appendix C, *Textsw Action Attributes*, provides the ACTION_* attributes that are available for use with a client-supplied notify procedure. These attributes are not standard attributes and cannot be used with xv_create(), xv_get(), or xv_set().

### Font Conventions Used in This Manual

*Italic* is used for:
- UNIX pathnames, filenames, program names, user command names, and options for user commands.
- New terms where they are introduced.

*Typewriter Font* is used for:
- Anything that would be typed verbatim into code, such as examples of source code and text on the screen.
- XView packages.*
- The contents of include files, such as structure types, structure members, symbols (defined constants and bit flags), and macros.
- XView and Xlib functions.
- Names of subroutines of the example programs.

*Italic Typewriter Font* is used for:
- Arguments to XView functions, since they could be typed in code as shown but are arbitrary.

*When referring to all members of a particular package, such as CANVAS, the notation CANVAS_* will be used. This should not be interpreted as a C-language pointer construct.*
Helvetica Italic is used for:

- Titles of examples, figures, and tables.

**Boldface** is used for:

- Sections and headings.

## Related Documents

*The C Programming Language* by B. W. Kernighan and D. M. Ritchie.

The following documents are included on the X11 source tape:

- OPEN LOOK Graphical User Interface Functional Specification
- OPEN LOOK Graphical User Interface Style Guide

The following books in the X Window System series from O'Reilly & Associates, Inc. are currently available:

- Volume One — Xlib Programming Manual
- Volume Two — Xlib Reference Manual
- Volume Three — X Window System User's Guide
- Volume Four — X Toolkit Intrinsics Programming Manual
- Volume Seven — XView Programming Manual
- Volume Eight — X Administrator’s Guide
- Quick Reference — The X Window System in a Nutshell
- PHIGS Programming Manual
- PHIGS Reference Manual
- Pexlib Programming Manual
- Pexlib Reference Manual
Requests for Comments

Please write to tell us about any flaws you find in this manual or how you think it could be improved, to help us provide you with the best documentation possible.

Our U.S. mail address, phone numbers, and e-mail addresses are as follows:

O’Reilly & Associates, Inc.
103 Morris Street, Suite A
Sebastopol, CA 95472
in USA 1-800-338-6887,
international +1 707-829-0515

UUCP: uunet!ora!xview Internet: xview@ora.com

Acknowledgements

The initial material for this manual was the appendices to the previous version of the XView Programming Manual by Dan Heller. This material has been considerably expanded and should be a helpful addition to the XView documentation.

Thanks to John Stone for all his help in preparing this manual. John dealt with formatting issues and checked and re-checked all the new attributes. Special thanks also go to Darci Chapman and Jon Lee for their assistance.

Dale Dougherty kept the big picture in sight and managed this project. Thanks to Lenny Mueller, Stephen Spainhour, Kismet McDonough, Chris Reilley, and the rest of the production and graphics team at O’Reilly and Associates who put the final manual together.

The XView developers at Sun Microsystems spent many hours going over many versions of this manual. In particular, Chris Kasso added extensively to this material and answered numerous questions. Thanks also go to Isa Hashim and Mitch Jerome for their comments and additions. Darren Austin, Sri Atreya, Shirley Joe, and Shanmugh Natarajan reviewed this manual and helped in numerous ways. Thanks also go to Tony Hillman, Bhaskar Prabhala, Greg Kimura, and Stan Raichlen. Despite all the reviews, any errors that remain are my own.

– Thomas Van Raalte
This page intentionally left blank

to preserve original page counts.
This page intentionally left blank

to preserve original page counts.
XView Package Summary

This section provides a brief introduction to XView and then goes on to introduce each of the XView packages. The packages are described in alphabetical order; their order in this section does not correspond to how they are used in any particular application. Refer to the XView Programming Manual for a general introduction to XView and for details on creating and working with XView objects.

The last part of this section provides a description of the Notifier. The Notifier maintains the flow of control in an application. The Notifier controls XView’s notification-based event handling using callback procedures supplied by application programmers. Also refer to the XView Programming Manual for more information on the Notifier.

1.1 XView and OPEN LOOK

XView provides a set of windows that include:

- Canvases on which programs can draw.
- Text subwindows with built-in editing capabilities.
- Panels containing items such as buttons, choice items, and sliders.
- TTY subwindows that emulate character-based terminals.

These windows are arranged as subwindows within frames, which are themselves windows. Frames can be transitory or permanent. Transient interactions with the user can also take place in menus which can pop up anywhere on the screen.

An important feature of the XView Toolkit is that it implements the OPEN LOOK Graphical User Interface (GUI). The OPEN LOOK GUI provides users with a simple, consistent, and efficient interface.

OPEN LOOK is supported by Sun and AT&T as the graphical user interface standard for System V Release 4. Users and developers benefit from a standard because it ensures consistent behavior across a number of diverse applications. Programmers can concentrate on the design of the application without having to “invent” a user interface for each application. XView was built based entirely on OPEN LOOK specifications that could be mapped easily into the X Window System.
The visual design of OPENLOOK is restrained. The design of each component in the user interface is simple and uncomplicated. The interface is based on a few simple concepts that result in a system that is easy to learn initially. And an XView application is relatively simple, and is easy to implement because many of the default values of interface components work well for most applications.

The definitive document on OPENLOOK for application programmers is the OPENLOOK Graphical User Interface Style Guide.

1.2 XView Packages

XView defines classes of objects in a tree hierarchy. For example, frame is a subclass of the more general class window, which in turn is a subclass of drawable. Drawable, like user interface object classes, is a subclass of the Generic Object class. Figure 1-1 shows the XView class hierarchy. Each class has identifying features that make it unique from other classes or packages. In XView, a class is often called a package, meaning a set of related functional elements. However, there are XView packages that are not members of the object class hierarchy, such as the Notifier package.

![Figure 1-1. XView class hierarchy](image)

XView Reference Manual
Some objects are visual and others are not. Visual objects include windows, scrollbars, frames, panels, and panel items, among others. Nonvisual objects are objects which have no appearance, per se, but they have information which aids in the display of visual objects. Examples of nonvisual objects include the server, screen, and font objects. The screen, for example, provides information such as the type of color it can display or the default foreground and background colors that objects might inherit. The display can provide information about what fonts are available for objects that display text.

All objects, both visual and nonvisual, are a part of this object classing system. The system is extensible, so you can create new classes that might or might not be based on existing classes.

XView uses static subclassing and chained inheritance as part of its object-oriented model. All objects of a particular class inherit the properties of the parent class (also known as a superclass). The Generic Object \texttt{XV\_OBJECT} contains certain basic properties that all objects share. For example, the same object can appear in many places on the screen to optimize storage. To keep a record of this, the Generic Object maintains a reference count of its instances. Since all objects have an owner, the parent of the object is stored in a field of the generic part of the object. As the needs of an object get more specific to a particular look or functionality, lower-level classes define properties to implement it.

Each class contains properties that are shared among all instances of that object. For example, panels are a part of the \texttt{PANEL} package, which has properties that describe, among other things, its layout (horizontal or vertical) or the spacing between items (buttons) in the panel. All panels share these properties, even though the state of the properties might differ for each instance of the object.

As mentioned earlier, XView uses subclassing so that each package can inherit the properties of its superclass. The \texttt{PANEL} package is subclassed from the \texttt{WINDOW} package, which has properties specific to all windows, such as window dimensions, location on the screen, border thickness, depth, visual, and colormap information. The \texttt{WINDOW} package is subclassed from the root object \texttt{XV\_OBJECT}, as are all objects, and the panel can access generic information such as the size and position of itself.

### 1.2.1 Object Handles

When you create an object, an XView function returns a handle for the object. Later, when you wish to manipulate the object or inquire about its state, you pass its handle to the appropriate function. This reliance on object handles is a way of information-hiding. The handles are opaque in the sense that you cannot see through them to the actual data structure which represents the object.

Each object type has a corresponding type of handle. Since C does not have an opaque type, all the opaque data types mentioned above are typedef’d to the XView type \texttt{Xv\_opaque} or \texttt{Xv\_object}. In addition to the opaque data types, there are several typedefs that refer not to pointers but to structures: \texttt{Event}, \texttt{Rect}, and \texttt{Rectlist}. Generally, pointers to these structures are passed to XView functions, so they are declared as \texttt{Event *}, \texttt{Rect *}, etc. The reason that the asterisk (*) is not included in the typedef is that the structures are publicly available.
1.2.2 Attribute-based Functions

A model such as that used by XView, which is based on complex and flexible objects, presents the problem of how the client is to manipulate the objects. The basic idea behind the XView interface is to provide a small number of functions, which take as arguments a large set of attributes. For a given call to create or modify an object, only a subset of the set of all applicable attributes will be of interest.

1.2.3 Creating and Manipulating Objects

There is a common set of functions that allows the programmer to manipulate any object by referencing the object handle. The functions are listed in Table 1-1.

Table 1-1. Generic Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>xv_init()</td>
<td>Establishes the connection to the server, initializes the Notifier and the Defaults/Resource-Manager database, loads the Server Resource Manager database, and parses any generic toolkit command-line options.</td>
</tr>
<tr>
<td>xv_create()</td>
<td>Creates an object.</td>
</tr>
<tr>
<td>xv_destroy()</td>
<td>Destroys an object.</td>
</tr>
<tr>
<td>xv_find()</td>
<td>Finds an object that meets certain criteria; or if the object doesn’t exist, creates it.</td>
</tr>
<tr>
<td>xv_get()</td>
<td>Gets the value of an attribute.</td>
</tr>
<tr>
<td>xv_set()</td>
<td>Sets the value of an attribute.</td>
</tr>
</tbody>
</table>

Using these six routines, objects can be created and manipulated from all packages available in XView. Table 1-2 lists the XView packages. Each of these packages is introduced in this section.

Table 1-2. XView Packages

<table>
<thead>
<tr>
<th>CANVAS</th>
<th>NOTICE</th>
<th>PANEL_NUMERIC_TEXT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMS</td>
<td>PANEL</td>
<td>PANEL_SLIDER</td>
</tr>
<tr>
<td>CURSOR</td>
<td>PANEL_ITEM</td>
<td>PANEL_TEXT</td>
</tr>
<tr>
<td>DRAGDROP</td>
<td>PANEL_BUTTON</td>
<td>SCREEN</td>
</tr>
<tr>
<td>DROP_SITE_ITEM</td>
<td>PANEL_CHOICE</td>
<td>SCROLLBAR</td>
</tr>
<tr>
<td>FONT</td>
<td>PANEL_CHECK_BOX</td>
<td>SELECTION</td>
</tr>
<tr>
<td>FRAME</td>
<td>PANEL_DROP_TARGET_ITEM</td>
<td>SERVER</td>
</tr>
<tr>
<td>FULLSCREEN</td>
<td>PANEL GAUGE</td>
<td>TEXTSW</td>
</tr>
<tr>
<td>ICON</td>
<td>PANEL_LIST</td>
<td>TTYSW</td>
</tr>
<tr>
<td>MENU</td>
<td>PANEL_MESSAGE</td>
<td>WINDOW</td>
</tr>
<tr>
<td>MENU_ITEM</td>
<td>PANEL_MULTILINE_TEXT</td>
<td></td>
</tr>
</tbody>
</table>
1.3 The CANVAS Package

A canvas is the area in which an application displays graphics and handles its input. An XView canvas object allows the user to view a graphic image that is similar to a painter’s canvas. This image may be too large for the window or even too large for the display screen. The viewable portion of the graphic image is part of image’s viewport or view window. Many different views of an image can use the same canvas object. While each view maintains its own idea of what it is displaying, the canvas object manages all the view windows as well as the graphic image that all views share. The ability for the canvas to maintain different views of the graphic image is a property that is inherited from the canvas’s superclass, the OPENWIN package. These properties provide for splitting and scrolling views. You cannot create a canvas object with multiple views; views are split and joined generally by the user via the attached scrollbars. It is possible to programmatically split and scroll views, but OPEN LOOK’s interface specification indicates that scrollbars provide the ability to split views. When a view is split, each new view may be further split into two more views, and so on. All the views are still a part of the same canvas object.

There are three types of windows involved with the canvas object:

Canvas Subwindow  Owned by a frame and manages one or more views. The canvas is subclassed from the OPENWIN package so all Openwin attributes must be set to the instance of the canvas object.

View Window    Represents the visible portion of the paint window—whenever the paint window associated with a view window changes, it is reflected in the view window. If there is more than one view window, the views are tiled. Vertical and/or horizontal scrollbars can be attached to the view subwindow to allow the user to modify which portion of the paint window is displayed for that particular view. The size of the view window can vary among all the views. Only views can be split. No graphics or user events take place in this window.

Paint Window   Graphics and events (mouse/keyboard) take place in the paint window. There is one paint window per view window. All paint windows in the canvas are the same size regardless of the size of the canvas or of the corresponding view windows. When a view is split, the old view reduces in size and a new view is created. With the new view, a new paint window is created that is identical to the paint window from the old view. This includes the same visual, width, height, depth and graphic image. However, callback functions and event masks are not inherited and must be manually installed in all new paint windows.

The CANVAS package is defined in the header file <xview/canvas.h> so programs that use canvases must include this file. This header file includes the OPENWIN package automatically. The owner of a canvas must be a FRAME object.
1.4 The CMS Package

The X Window System has various ways of allocating, specifying, and using colors. While all of these methods are available to applications without XView intervening, XView provides its own model for color specification that may be used as an alternative. It does not provide anything more than what is already available, but it may provide a simpler interface to request and specify colors. This model is especially useful when specifying colors for XView objects, such as panel buttons and scrollbars.

XView applications deal with color by using colormap segments. Window-based objects (canvases, panels, textsw, etc.) use colormap segments to get their colors. These objects get a default colormap segment when they are created, but you can assign a new one using the \texttt{WIN\_CMS} attribute. Colormap segments must be applied to windows to assure that the window can access the color you are attempting to draw into.

A colormap segment from the CMS package is a subset of the available cells in a colormap on the X server. These are XView entities (i.e., not Xlib) that provide a veneer over the Xlib color mechanism. Colormap segments can be created as either static or dynamic and are derived from an underlying colormap of the same type.

Applications that use color must include the file <\texttt{xview/cms.h}>. The owner of a colormap segment is the XView screen object with which the colormap is associated. If an owner is not specified (\texttt{NULL} owner), the default screen of the default server is used as the owner.

1.5 The CURSOR Package

A cursor is an image that tracks the mouse on the display. Each window has its own cursor which you can change. There are some cursors defined by OPENLOOK that correspond to specific window manager operations such as resizing or dragging windows. For these cases, you cannot redefine a cursor. However, for windows in your application, you can assign any cursor image you like.

To use the \texttt{CURSOR} package, include the header file <\texttt{xview/cursor.h}>. The owner of the cursor may be any XView object. The root window associated with the XView object is used internally by the \texttt{CURSOR} package. If the owner is \texttt{NULL}, then the root window of the default screen is used.

A number of predefined cursors are available in the \texttt{CURSOR} package for use as OPENLOOK cursors. To use these cursors, you may specify the CURSOR\_SRC\_CHAR and CURSOR\_MASK\_CHAR attributes with certain predefined constants as values for these attributes. There are some OPENLOOK cursor \texttt{defines} prefixed by OLC\_ in <\texttt{xview/cursor.h}>.

The hotspot on a cursor is the location in which the cursor is located if the user generates an event like pressing a mouse button or typing at the keyboard, or if you were to query its position. For example, if a cursor is shaped like an arrow, the hotspot should be at the tip of the arrow. If the hotspot for a cursor were set to (0, 0) then the hotspot would be the upper-left corner of the image used. A cursor shaped like a bull’s eye (16x16) might have its hotspot at (7, 7) to indicate that the focus for the cursor is in the middle.
1.6 The DRAGDROP and DROP_SITE_ITEM Packages

Drag and drop operations are facilitated using two packages: the DRAGDROP package and the DROP_SITE_ITEM package. DRAGDROP is subclassed from the SELECTION_OWNER package and represents the source of the drag. DROP_SITE_ITEM is subclassed from the Generic package. A drop-site item indicates a destination that is a valid drop-site. A valid drop-site is a region that may either preview a drop or receive a drop. A drag and drop operation, such as the familiar procedure of dragging a file from a folder and dropping it into another folder, allows you to easily transfer data.

Applications need to include the file <xview/dragdrop.h> to use these packages. Drag and drop supports drop previewing where the drop-site image changes to show that it is a valid drop-site and drag feedback where the pointer image (cursor) changes to indicate that an item is being dragged.

Attributes for DRAGDROP use a DND_ prefix. Attributes for DROP_SITE_ITEM use a DROP_SITE prefix. The owner of either a DRAGDROP object, or a DROP_SITE_ITEM object is a window.

1.7 The FONT Package

In X, a large number of fonts are provided on the server. Deciding which font to use and then trying to specify fonts by name can be difficult since there are many different styles and sizes of fonts. Most fonts are used to render text strings. So the images, or glyphs, represent a character set-defined language used. However, a font may be built to support glyphs for other languages or to provide a set of glyphs. Fonts are stored on the server and are associated with the display of your workstation. The font ID is stored in the graphics context (GC), which is used by Xlib functions like XDrawString(). Using fonts to render text is perhaps the most common application. For example, the Courier font family displays the classic typewriter or constant-width character set. This text is set in Times-Roman, a proportionally spaced font. Often within a font family, there are different styles, such as bold or italic, and different point sizes.* For example, Helvetica bold 14 refers to the Helvetica font family; bold is the style and 14 is the point size.

Not all server fonts have a variety of styles and sizes. These special-purpose fonts are generally specified by name only—there are no corresponding styles or families for these fonts.

When accessing fonts, you typically want to specify a font either by name or by the family, style, and size or scale of the font. In addition, XView provides an interface for determining the dimensions (in pixels) of characters and strings rendered in a specified font.

OPEN LOOK uses predefined fonts for certain items such as panel buttons and other user interface elements. These items cannot be changed, but you can assign text fonts to panel choices, text subwindows and other types of windows.

*Note that point sizes on workstations are based on pixels, whereas point sizes for typesetters and printers are based on inches.
Applications that use the \texttt{FONT} package must include the header file, \texttt{<xview/font.h>}. In XView, when a font object is created, it loads the font from the X server. When we say, “create a font,” we really mean, “load a font from the server and create an XView font object associated with that font.”

While fonts can be created using \texttt{xv\_create()}, it may not be necessary to create a new instance of a font. Fonts are typically cached on the server, and XView may already have a handle to a particular font. Therefore, you would obtain a handle to the font, if it already exists, rather than open another instance of the same font. \texttt{xv\_find()} can be used to return the handle of an existing font. If the handle does not exist, \texttt{xv\_find()} can create a new instance of the font.

The owner of the font is usually the window in which the font is going to be used. The actual X font is loaded from the server associated with the owner object. If the owner is \texttt{NULL}, the default server is used. Fonts may be used on any window, or in memory pixmaps, or a server image (a \texttt{Server\_image} object), but these objects must have the same display (Server) associated with them as the font, or you will get an X Protocol error. What this means, is that a font can only be used on the server on which it was created.

\section*{1.8 The FRAME Package}

A frame is a container for other windows. It manages the geometry and placement of subwindows that do not overlap and are fixed within the boundary of the frame. The OPENLOOK specification refers to subwindows, or panes, as tiled windows because they do not overlap one another. Subwindow types include canvases, text subwindows, panels, and scrollbars. These subwindows cannot exist without a parent frame to manage them.

The \texttt{FRAME} package provides the following capabilities:

- A communication path between the application and the window manager.
- A mechanism to receive input for the application.
- A visual container for user interface objects.
- A method to group windows with related functionality.
- A mechanism to manage footers.

A frame depends upon the window manager for its decorations and many basic operations. The \texttt{FRAME} package does not manage headers (title bars), resize corners, or the colors of those objects. These are all strictly functions of the window manager. The application gives hints to the window manager about some of these attributes through the \texttt{FRAME} package (including not to display decorations at all if so desired), but results vary depending on which window manager the user is running.
Frames do not manage events; this task is left up to the windows that the frame manages. That is, frames do not get mouse and keyboard events and propagate them to child windows. While frames are subclassed from the window package, the frame’s window rarely sees any events at all, and if they do, these are not intended to be processed by the application programmer.

Basically, two types of frames are available in XView: base frames and command frames. The main frame of the application is called the base frame. The base frame resides on the root window; its handle is passed to `xv_main_loop()` to begin application processing.

A special kind of frame, called a command frame, is created with a panel subwindow by default. Command frames are useful as help frames, property frames, and such defined by OPEN LOOK. Programmatically, a command frame is no different from a frame with one subwindow that is a panel and a pushpin if run under the olwm (OPEN LOOK window manager).

A base frame’s parent or owner, is the root window, whereas a subframe’s parent is another frame (either a base frame or a subframe). When a frame goes away (quit or close), all of its child windows, including subframes, also go away. For example, assume you create a command subframe to display application-specific help. When this command subframe is activated, it might display explanatory text along with an OK button to dismiss the help. If you close the base frame, the help subframe also closes.

XView allows for multiple frames that are not children of the base frame. For instance, you could create a help frame that is independent of the application’s base frame. The parent of this frame is the root window of the display and not the base frame. The help frame will remain visible even if the base frame goes away. The term subframe defines a relationship among frames at creation time and a slight difference in functionality.

### 1.9 The FULLSCREEN Package

The FULLSCREEN package is used to grab the X server, and an instance of it is considered a nonvisual object. Nonvisual objects are not viewed on the screen, but they have a place in the XView object hierarchy. Nonvisual objects are typically used internally by XView and are seldom used directly in an application. Working with FULLSCREEN objects may not be essential to all programmers.
1.10 The ICON Package

A user may close an application to save space on the display. The program is still running and it may even be active, but it is not receiving input from the user. In order to represent the application in its closed state, an icon is used. An icon is a small picture that represents the application.

The graphic image that icons use may be used for other purposes and, therefore, may be shared among other objects in the application. But the icon image should be designed to easily identify the application while in a closed state. Icons may also have text associated with them. Space is limited, so the text is usually the name of the application.

To use the ICON package, include the header file `<xview/icon.h>`. The owner of an icon is a base frame, but it may be created with a NULL owner. Once an icon is assigned to a frame, the owner of the icon is changed to that frame. This is an example of delayed binding.

When destroying an icon, the server image associated with the icon is not destroyed— it is the application programmer’s responsibility to free the server image and the pixmap associated with the icon if needed.

1.11 The MENU and MENUITEM Packages

Menus play an important role in an application’s user interface. An OPEN LOOK menu may display text or graphics. Menus may be attached to most XView objects such as menu buttons, scrollbars, or text subwindows, or they may exist independently from objects and be displayed on demand.

The user may cause a menu to be pinned up by selecting an optional pushpin in the pop-up menu. When this happens, the menu is taken down and a corresponding command frame is put up at the same location. Panel items in the pinup window correspond to the menu items in the menu. Once a menu has been pinned up, the user continues to interact with it just as if the menu were popped up each time. Menus that are used frequently are good candidates for having pushpins so the user does not have to repeat the sequence of redisplaying the menu to make selections.

OPEN LOOK requires that menus have titles. Menus or submenus that originate from menu buttons or pullright items do not need to have titles, since the name of the menu button or menu item acts as the title.

Fonts may not be specified in either menu items or menu titles; menu items follow the same constraints outlined for panel buttons. However, if text is not used, then menu items may contain graphic images, in which case, the font is of no concern. That is, you could specify a Server_image that has a string rendered in a particular font.
1.11.1 Menu Types

There are three different types of menus: *pop-up*, *pulldown*, and *pullright* menus. The general term pop-up menu may describe all three types in certain contexts since menus are *popped up*. However, pulldown and pullright menus have distinct characteristics that make them unique.

Pop-up Menus Pop-up menus are displayed when the user selects the MENU mouse button over XView objects such as *scrollbars* or *text subwindows*. An OPEN LOOK window manager also utilizes pop-up menus in the root window and from base frame title bars. XView objects handle the display of menus automatically.

Pulldown Menus Pulldown menus are attached to *menu buttons*. Menu buttons have a set of choices associated with them that the user can access only via the pulldown menu. When the user presses the MENU mouse button over a menu button, the choices are displayed in the form of a pulldown menu. If the menu button is selected using the SELECT button, the default menu item is selected.

Pullright Menus OPEN LOOK provides for items in the menu to have pullright menus associated with them. Also called *cascading menus*, these menus are activated from the user dragging the MENU mouse button to the right of a menu item that has an arrow pointing to the right. The cascading menu that results is a pop-up menu that can also have menu items with pullrights attached.

1.11.2 Menu Items

In addition to the menu types, there are different types of menu items: *choice*, *exclusive*, and *nonexclusive*. The different menu item types may be associated with each type of menu.

Each menu has a *default selection* associated with it. This item is displayed uniquely from other menu items and designates a default action to take if the user wants to select the menu without displaying it (see *pulldown menus* above). Typically, the 0th item in the menu is the default, but that may be changed either by the application or by the user.

Choice Items The choice item is the default menu item type used when a menu is created. The default selection in a menu has a ring around it. When a pop-up menu is displayed, it is positioned so that the mouse is pointing at the default item. Choice menu items may have pullright menus associated with them, in which case there is a pullright arrow at the right side of the item. If the selection of a menu item brings up a dialog box (command frame), then the label for the menu item typically ends in ellipses (…).

Exclusive Items When a choice item is selected, an action is taken and the menu forgets about it. Exclusive menu items retain the fact that they are selected even after the menu has popped down. If the user selects a new item,
the new item is remembered. Because this is an exclusive menu, only one choice may be selected at a time. The default item is indicated by a double-lined box around the item.

When exclusive settings are used on menus, the current choice has a bold border when the pointer is not on a menu choice. When the user drags the pointer onto other settings, the bold border follows the pointer. Exclusive choice menus may not have items with pullright menus.

Nonexclusive Items Also called toggle items, menus that have toggle items support multiple choices from the menu to be selected at the same time. That is, the user may toggle whether a particular choice is selected. This action has no affect on the other menu items.

The MENUITEM package allows you to create separate menu items using separate calls to xv_create(). The attributes used are menu item-specific attributes—the same as those that are used for a menu’s MENU_ITEM attribute.

1.12 The NOTICE Package

A notice is a pop-up window that notifies the user of a problem or asks a question that requires a response. Generally, notices report serious warnings or errors. OPEN LOOK notices do not have headers or footers and cannot be moved.

XView defines two types of notices, standard notices and screen-locking notices:

- Standard notices do not lock the screen and are placed centered in the “owner” frame. This type of notice may either block the application’s thread of execution, or not block.

- Screen-locking notices lock the screen and block the thread of execution for all applications (the screen is locked with X grabs). These notices appear with a shadow that emanates from the location where an action in an application initiates the notice. This may be a panel button, such as “Quit”, or some other XView object.

To use the NOTICE package, include the header file <xview/notice.h>. It provides the necessary types and definitions for using the package. A notice object’s type is Xv_Notice.
1.13 The PANEL Package

The PANEL package implements the OPEN LOOK control area. Panels are used in many different contexts—property sheets, notices, and menus all use panels in their implementation. The main function of a panel is to manage a variety of panel items. Because some panel items may not contain windows that handle their own events, the PANEL package is responsible for propagating events to the appropriate panel item.

Panels set up and manage their own event handling masks and routines for themselves and their panel items. The application does not set event masks or install an event callback routine unless it needs to track events above and beyond what the PANEL package does by default (typical applications will not need to do this). The PANEL package handles all the repainting and resizing events automatically. Panels are not used to display graphics, so there is no need to capture repaint events. Rather than deal with other events specifically, callback routines are not installed on panels, but set for each panel item. Because of the varying types of panel items, each item’s callback function may be invoked by a different action from the user. While clicking on a panel button is all that is necessary to activate the button’s callback routine, a text panel item might be configured to call its notification callback routine upon the user pressing the RETURN key.

Since panel items express interest in different events, it is the responsibility of the PANEL package to track all events within the panel’s window and dispatch events to the proper panel item depending on its type. In some cases, if an event happens over a certain panel item and that item is not interested in that event, the event may be sent to another panel item. For example, what happens if a key is pressed over a panel button? Because the panel button has no interest in the event, the panel will send the event to a text panel item, if one exists elsewhere in the panel.

A panel’s owner is a frame. All programs that use panels or panel items must include `<xview/panel.h>.

1.13.1 Panel Items

The user interacts with items through various methods ranging from mouse button selection to keyboard input. This interaction typically results in a callback function being called for the panel item. The callback functions also vary on a per-item basis. Each item type is described in the following sections.

1.13.1.1 Button Items

A button item allows the user to invoke a command or bring up a menu. The button’s label identifies the name of the command or menu. A button label that ends in three dots (…) indicates that a pop-up menu will be displayed when the button is selected.
There are several types of panel button items:

- Panel Buttons (shown in Figure 1-2)
- Menu Buttons (shown in Figure 1-3)
- Abbreviated Menu Buttons (shown in Figure 1-4)

![Figure 1-2. Panel item created with PANEL_BUTTON package](image)

![Figure 1-3. Panel button with an attached menu](image)
1.13.1.2 Choice Items

Choice items provide a list of different choices to the user in which one or more choices may be selected. There are variations of choice items which implement different OPEN LOOK objects such as:

- Exclusive and Nonexclusive Choices (or Settings)
- Abbreviated Choice Items
- Checkboxes

The figures below show several, but not all of the different types of choice items.
1.13.1.3 Drop Target Items

A panel drop target item is a bordered image in a panel area that is used to transfer data to or from applications. Before you use a panel drop target item you need to be familiar with the SELECTION and DRAGDROP packages.

A panel drop target item is an object in the class Panel_drop_target_item which is equivalent to a Panel_item. A drop target item’s owner is a Panel.

1.13.1.4 Panel Extension Items

Panel extension items support additional user defined panel items. Use of this type of panel item is an advanced topic that is covered in Chapter 25, XView Internals, in the XView Programming Manual.
1.13.1.5 Gauges

Gauges are just like sliders, but they are “output only” items. That is, you set the value of the item and the display of the gauge changes just as it would for sliders. Also, there is no optional type-in field and there is no slider bar for the user to interactively change the value of the gauge. The gauge is intended to be used only as a feedback item.

To create a gauge, use the PANEL_GAUGE package.

![Figure 1-9. Panel drop target items](image)

![Figure 1-10. Panel item created with the PANEL_GAUGE package](image)
1.13.1.6 List Items—Scrolling Lists

OPEN LOOK’s specification for *scrolling lists* is implemented by the PANEL_LIST panel item. List items allow the user to make selections from a scrolling list of choices larger than can be displayed on the panel at one time. The selections can be exclusive or nonexclusive, like the choice items outlined in the previous section. The list is made up of strings or images and a scrollbar that functions like any scrollbar in XView, except that it cannot be split.

![List Item](image)

*Figure 1-11. Panel list item created with the PANEL_LIST package*

1.13.1.7 Message Items

Message items display a text or image message within a panel. The only visible component of a message item is the label itself. Message items are useful for annotations of all kinds, including titles, comments, descriptions, pictures, and dynamic status messages. The message is often used to identify elements on the panel. A message has no value.

![Message Item](image)

*Figure 1-12. Panel item created with PANEL_MESSAGE package*
1.13.8 Multiline Text Items

Multiline text items are a special type of panel text item that allow a text field containing multiple lines.

1.13.9 Numeric Text Items

Panel numeric text items are virtually the same as panel text items except that the value displayed is of type `int`. Also, convenience features (such as increment and decrement buttons) ease the manipulation of the text string’s numeric value, but there is little programmatic difference between the text item and the numeric text item. You can create a numeric text item using the `PANEL_NUMERIC_TEXT` package.
1.13.1.10 Slider Items

Slider items allow the graphical representation and selection of a value within a range as shown in Figure 1-16. Sliders are appropriate for situations where it is desired to make fine adjustments over a continuous range of values. The user selects the slider bar and drags it to the value that he wishes. A slider has the following displayable components: the label, the current value, the slider bar, and the minimum and maximum allowable integral values (the range), end boxes, tick marks, tick mark minimum and maximum tick strings, as well as minimum and maximum value text strings.

1.13.1.11 Text Items

A panel text item contains as its value a NULL-terminated string. Typically, it contains only printable ASCII characters with no newlines. When a panel receives keyboard input (regardless of where the pointer is as long as it is within the boundaries of the panel), the keyboard event is passed to the item with the keyboard focus. A caret is used to indicate the insertion point where new text is added. You can type in more text than fits on the text field. If this happens, a right arrow pointing to the left will appear on the left on the field, indicating that some text to the left of the displayed text is no longer visible. Similarly, if text is inserted causing text on the right to move out of the visible portion of the text item, then an arrow pointing to the right will appear to the right of the text.
1.13.2 Scrollable Panels

Scrollable panels are not OPEN LOOK-compliant, but are provided for historical reasons. They are basically just like panels, except that typically not all panel items are in view. A vertical scrollbar attached to the panel allows the user to navigate to the panel items desired. Again, because this type of interface is not OPEN LOOK-compliant, you are discouraged from using this package.

Scrollable panels are created the same way panels are, but the package name to use is `SCROLLABLE_PANEL`. However, the scrollable panel package does not create the scrollbars, you must create them separately.

1.14 The SCREEN Object

An `Xv_Screen` is associated with virtually all XView objects. To use the `Xv_Screen` object, you must include the file `<xview/screen.h>`. The `Xv_Screen` object carries useful information such as the screen number of the root window, all the visuals, the colormap, the server and so on, that are associated with that screen.

The `Xv_Screen` object differs from the `Screen` data structure defined by Xlib and, in fact, has nothing to do with the X11 `Screen` data type (defined in `<X11/Xlib.h>`).
1.15 The SCROLLBAR Package

Scrollbars are used to change what you view in a subwindow. For instance, in a text subwindow, scrollbars are used to scroll through a document. In a canvas subwindow, scrollbars can be used to see another portion of the paint window (which can be larger than the canvas subwindow).

The definitions necessary to use scrollbars are found in the header file <xview/scrollbar.h>. The owner must be an object subclassed from the OPENWIN package or the FRAME package. The scrollbar inherits certain attributes from the parent while other attributes are initialized automatically. For example, if the owner of the scrollbar is a canvas, the scrollbar’s color is inherited from the canvas, while the scrollbar’s object length is set by the canvas explicitly; that is, you are not required to set it. This is usually desirable when creating objects that are used together.

1.16 The SELECTION Package

The X Window System provides several methods for applications to exchange information with one another. One of these methods is the use of the selections. A selection transfers arbitrary information between two clients. XView Version 3 provides a selection mechanism that is implemented using the SELECTION and SELECTION_ITEM packages. The selection package and its sub classes, including: the SELECTION_REQUESTOR package and the SELECTION_OWNER package, allow data to be move between applications or within an application. These packages replace the selection service used in previous versions of XView, which required special functions and structures. The old selection service is still supported; it is described in Appendix A, The Selection Service, of the XView Programming Manual.

1.17 The SERVER Package

The SERVER package may be used to initialize the connection with the X server running on any workstation on the network. Once the connection has been made, the package allows you to query the server for information. xv_init(), the routine that initializes the XView Toolkit, opens a connection to the server and returns a handle to an Xv_Server object. While more than one server can be created, xv_init() only establishes a connection to one server. The server object returned by xv_init() is also the server pointed to by the external global variable, xv_default_server. Programs that do not save the Xv_Server object returned by xv_init() can reference this global variable instead.

Subsequent connections to other X11 servers must be made using separate calls to xv_create(). Note that using separate screens is not the same as establishing a connection to other servers—the same server can support multiple screens.
When making any reference to Xv_Server objects, applications should include <xview/server.h>. There is no owner for a server, the owner parameter is ignored and you may pass NULL.

1.18 The TEXTSW Package

This TEXTSW package allows a user or client to display and edit a sequence of ASCII characters. A text contains a vertical scrollbar but may not contain a horizontal scrollbar. The vertical scrollbar can be used to split views into several views. The font used by the text can be specified using the TEXTSW_FONT attribute, but only one font per text subwindow can be used, regardless of how many views there may be.

The contents of a text subwindow are stored in a file or in memory on the client side, not on the X server. Whether the source of the text is stored on disk or in memory is transparent to the user. When the user types characters in the text subwindow, the source might be changed immediately or synchronized later depending on how the text subwindow is configured. The TEXTSW package provides basic text editing features such as inserting arbitrary text into a file. It also provides complex operations such as searching for and replacing a string of text.

Applications need to include the file <xview/textsw.h> to use text subwindows.

1.19 The TTYSW Package

The TTY (or terminal emulator) subwindow emulates a standard terminal, the principal difference being that the row and column dimensions of a tty subwindow can vary from that of a standard terminal. In a tty subwindow, you can run arbitrary programs, including a complete interactive shell. Or you can emulate terminal interface applications that use the curses(3X) terminal screen optimization package without actually running a separate process. The TTY subwindow accepts the standard ANSI escape sequences for doing ASCII screen manipulation, so you can use termcap or termio screen-handling routines.

Programs using tty subwindows must include the file <xview/tty.h>. The default tty subwindow will fork a shell process and the user can use it interactively to enter commands. This program does not interact with the processing of the application in which the TTY subwindow resides; it is an entirely separate process.
1.20 The Notifier

The Notifier maintains the flow of control in an application. To understand the basic concepts of the Notifier, we must distinguish between two different styles of input handling, *mainline* input and *event-driven* input, and consider how they affect where the flow of control resides within a program.

1.20.1 Mainline Input Handling

The traditional type of input handling of most text-based applications is mainline-based and input-driven. The flow of control resides in the main routine and the program *blocks* when it expects input. That is to say, no other portion of the program may be executed while the program is waiting for input. For example, in a mainline-driven application, a C programmer will use `fgets()` or `getchar()` to wait for characters that the user types. Based on the user’s input, the program chooses an action to take. Sometimes, that action requires more input, so the application calls `getchar()` again. The program does not return to the main routine until the processing for the current input is done.

The tight control represented by this form of input handling is the easiest to program since you have control at all times over what to expect from the user and you can control the direction that the application takes. There is only one source of input—the keyboard—and the user can only respond to one interface element at a time. A user’s responses are predictable in the sense that you know that the user is going to type *something*, even if you do not know what it is.

1.20.2 Event-driven Input Handling

Windowing systems are designed such that many sources of input are available to the user at any given time. In addition to the keyboard, there are other input devices, such as the mouse. Each keystroke and mouse movement causes an *event* that the application might consider. Further, there are other sources of events such as the window system itself and other processes. Another aspect of event-driven input handling is that you are not guaranteed to have any predictable sequence of events from the user. That is, a user can position the mouse on an object that receives text as input. Before the user is done typing, the user can move the mouse to another window and select a panel button of some sort. The application cannot (and should not) expect the user to type in window A first, then move to window B and select the button. A well-written program should expect input from any window to happen at any time.
1.20.3 Functions of the Notifier

The Notifier can do any of the following:

- Handle *software interrupts*—specifically, UNIX signals such as SIGINT or SIGCONT.
- Notice state changes in processes that your process has spawned (e.g., a child process that has died).
- Read and write through file descriptors (e.g., files, pipes and sockets).
- Receive notification of the expiration of timers so that you can regularly flash a caret or display animation.
- Extend, modify or monitor XView Notifier clients (e.g., noticing when a frame is opened, closed or about to be destroyed.)
- Use a non-notification-based control structure while running under XView (e.g., porting programs to XView).

The Notifier also has provisions, to a limited degree, to allow programs to run in the Notifier environment without inverting their control structure.

1.20.4 How the Notifier Works

When you specify *callbacks* or *notify procedures*, the XView object specified is said to be the *client* of the Notifier. Generally stated, the Notifier detects events in which its clients have expressed an interest and dispatches these events to the proper clients in a predictable order. In the X Window System, events are delivered to the application by the X server. In XView, it is the Notifier that receives the events from the server and dispatches them to its clients. After the client’s notify procedure processes the event, control is returned to the Notifier.

1.20.4.1 Restrictions

The Notifier imposes some restrictions on its clients. Designers should be aware of these restrictions when developing software to work in the Notifier environment. These restrictions exist so that the application and the Notifier do not interfere with each other. More precisely, since the Notifier is multiplexing access to user process resources, the application needs to respect this effort so as not to violate the sharing mechanism.

For example, a client should not call `signal(3)`. The Notifier is catching signals on behalf of its clients. If a client sets up its own signal handler, then the Notifier will never notice the signal. The program should call `notify_set_signal_func()` instead of `signal(3)`.
This page intentionally left blank

to preserve original page counts.
This page intentionally left blank

to preserve original page counts.
This section lists all XView attributes in alphabetical order. Each attribute’s description is in the format shown by the entry on this page. Only the fields appropriate to the particular attribute are shown.

### INTRODUCTION
This field provides a brief description of the attribute.

**Return Type:** The return type describes the type that the attribute should return on a call to `xv_get()`. Note that this return type is normally coerced by the programmer. This field is only shown when the attribute is valid with `xv_get()`.

**Argument:** This shows the first programmer supplied value associated with the attribute. If an attribute has multiple values, then the type of each value is shown in multiple Argument fields.

**Valid Values:** Shows a list (in brackets) of values that are valid for the attribute.

**Default:** The default field shows the default value for the attribute.

**Procs:** Shows the attribute’s valid procedures. In this field, `xv_create`, `xv_find`, `xv_get`, and `xv_set` exclude the `xv_` prefix.

**Objects:** The objects field shows the valid object or objects for the attribute. Attributes are divided into three classes: generic, common, and specific. (For more information, refer to the *XView Programming Manual*, Section 2.2.3, Types of Attributes.)

**Callback:**
```c
returntype
function_name_proc(arg1, arg2, arg3)
type1   arg1;
type2   *arg2;
type3   *arg3;
```

The callback field shows the application programmer defined callback procedure associated with the attribute. If there are multiple callbacks associated with the attribute, each callback has its own Callback field. The function’s return value is provided, along with a description of its arguments, if any.

**Usage:** If an example is provided it will be shown in this field.

**See Also:** Where there are numbers, they refer to chapters and sections in the *XView Programming Manual*, where the attribute is described (letters refer to appendices). A list of related attributes and/or other related information may also be shown.
**ATTR_LIST**

Specifies a NULL-terminated attribute-value list. It has no value type or default, but when used, it must be the first attribute in an attribute-value list. ATTR_STANDARD_SIZE, defined in *attr.h*, defines the maximum size of the attribute-value list.

Argument: Attr_avlist
Proc: create, set
Objects: All
Usage:

```c
xv_create(object, pkg,
      ATTR_LIST, avlist,
      other_attrs, ..., 0);
```

See Also: 25.2.2.1, attr_create_list() in Section 3, Procedures and Macros

**CANVAS_AUTO_CLEAR**

Same as the OPENWIN_AUTO_CLEAR attribute.

**CANVAS_AUTO_EXPAND**

If TRUE, canvas width and height are never allowed to be less than the edges of the canvas window.

Argument: Bool
Default: TRUE
Proc: create, get, set
Objects: Canvas
See Also: 5.4.1

**CANVAS_AUTO_SHRINK**

If TRUE, canvas width and height are never allowed to be greater than the edges of the canvas window.

Argument: Bool
Default: TRUE
Proc: create, get, set
Objects: Canvas
See Also: 5.4.1

**CANVAS_CMS_REPAINT**

Specifies whether the canvas repaint procedure is called whenever a new colormap segment is set on the canvas, and/or the foreground and background colors of the canvas are changed.

Argument: Bool
Default: FALSE
Proc: create, get, set
Objects: Canvas
See Also: 5.3

**CANVAS_FIXED_IMAGE**

Sets the BitGravity for the canvas paint windows to be NorthWestGravity when TRUE or ForgetGravity when FALSE.

Argument: Bool
Default: TRUE
Proc: create, get, set
Objects: Canvas
See Also: 5.2, WIN_BIT_GRAVITY, CANVAS_RESIZE_PROC

---

32 XView Reference Manual
CANVAS_HEIGHT
Specifies the height of the canvas paint window in pixels.
Argument: int
Default: 0
Procs: create, get, set
Objects: Canvas
See Also: 5.4.2, CANVAS_MIN_PAINT_HEIGHT, CANVAS_WIDTH, XV_RECT

CANVAS_MIN_PAINT_HEIGHT
Specifies the minimum height of the canvas’ paint window(s) in pixels. Any attempt to change the CANVAS_HEIGHT to be smaller than this value will have the effect of setting it to the value of CANVAS_MIN_PAINT_HEIGHT.
Argument: int
Default: 0
Procs: create, get, set
Objects: Canvas
See Also: 5.4.1, CANVAS_AUTO_EXPAND, CANVAS_AUTO_SHRINK, CANVAS_HEIGHT, CANVAS_WIDTH

CANVAS_MIN_PAINT_WIDTH
Specifies the minimum width of the canvas’ paint window(s) in pixels. Any attempt to change the CANVAS_WIDTH to be smaller than this value will have the effect of setting it to the value of CANVAS_MIN_PAINT_WIDTH.
Argument: int
Default: 0
Procs: create, get, set
Objects: Canvas
See Also: 5.4.1, CANVAS_AUTO_EXPAND, CANVAS_AUTO_SHRINK, CANVAS_HEIGHT, CANVAS_WIDTH

CANVAS_NO_CLIPPING
This attribute is for SunView compatibility. For more information, refer to the manual Converting SunView Applications.

CANVAS_NTH_PAINT_WINDOW
Returns the paint window associated with the nth view, and takes an argument of the value of n. 0 is the index of the first view. NULL is returned if the view does not exist.
Return Type: Xv_Window
Argument: int
Procs: get
Objects: Canvas
Usage:
Canvas canvas;
Xv_Window paint_window;
paint_window = (Xv_Window)xv_get(canvas, CANVAS_NTH_PAINT_WINDOW, 1);

The example above will return the second paint window of the canvas.
See Also: 5.6.3.2

XView Attributes
CANVAS_PAINT_CANVAS_WINDOW

Gets the canvas from a canvas paint window.

Return Type: Canvas
Procs: get
Objects: Canvas_paint_window
See Also: CANVAS_VIEW_CANVAS_WINDOW

CANVAS_PAINT_VIEW_WINDOW

Gets the canvas view from a given canvas paint window.

Return Type: Canvas
Procs: get
Objects: Canvas_paint_window
See Also: CANVAS_VIEW_PAINT_WINDOW

CANVAS_PAINTWINDOW_ATTRS

Distributes specified attribute values across all paint windows in a given canvas. It takes an in-line attribute-value list of window attributes.

Argument: A-V list
Procs: create, set
Objects: Canvas
Usage:

    xv_set(canvas, CANVAS_PAINTWINDOW_ATTR,
           WIN_EVENTPROC, canvas_event,
           WIN_FOREGROUND_COLOR, 0
           NULL,
           NULL);

See Also: 20.3

CANVAS_REPAINT_PROC

Names a procedure called when canvas paint window has been damaged and must be repaired (repainted).

Argument: void (*canvas_repaint_proc) ()
Default: NULL
Procs: create, get, set
Objects: Canvas
Callback: (Used when CANVAS_X_PAINT_WINDOW set to FALSE)

    void
    canvas_repaint_proc(canvas, paint_window, repaint_area)
    Canvas canvas;
    Xv_Window paint_window;
    Rectlist *repaint_area;

    canvas is the canvas that was damaged.
    paint_window is the window to repaint.
    repaint_area is a pointer to the list of Rects that is to be repainted.
Callback 2: 

(Used when CANVAS_X_PAINT_WINDOW set to TRUE)

When CANVAS_X_PAINT_WINDOW is TRUE, the CANVAS_REPAINT_PROC callback is called with the following parameters:

```c
void canvas_repaint_proc(canvas, paint_window, display, xid, xrectlist)
Canvas canvas;
Xv_Window paint_window;
Display *display;
Window xid;
Xv_xrectlist *xrects;
```

canvas is the canvas that was damaged.
paint_window is the window to repaint.
display is the display handle to the X11 server connection.
xid is the X11 window identifier for the canvas paint window.
The Xv_xrectlist structure, defined in `<xview/xv_xrect.h>`, contains an array of XRectangles and a count that specifies the repaint area for the canvas paint window.

See Also: 5.3, CANVAS_REPAINT_PROC, CANVAS_X_PAINT_WINDOW, WIN_EVENT_PROC

CANVAS_RESIZE_PROC

Names a procedure called when the canvas paint window’s width or height changes. Note that if CANVAS_FIXED_IMAGE is set to TRUE, which it is by default, the resize procedure will not be called when the canvas is resized smaller.

Argument: void (*canvas_resize_proc) ()
Default: NULL
Procs: create, get, set
Objects: Canvas
Callback:

```c
void canvas_resize_proc(canvas, width, height)
Canvas canvas;
int width;
int height;
```
canvas is the canvas being resized.
width and height are the new dimensions of the canvas.

See Also: 5.4.3, CANVAS_FIXED_IMAGE, CANVAS_REPAINT_PROC, WIN_BIT_GRAVITY

CANVAS RETAINED

Specifies whether the X server should attempt to retain backing store for the canvas paint windows. Note that this does not guarantee that the window is retained as the server may not be able to comply.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Canvas
See Also: 5.2

XView Attributes
**CANVAS_VIEW_CANVAS_WINDOW**

Gets the canvas from a canvas view window.

Argument: Canvas
Procs: get
Objects: Canvas_view
See Also: CANVAS_PAINT_CANVAS_WINDOW, CANVAS_VIEW_PAINT_WINDOW

**CANVAS_VIEW_PAINT_WINDOW**

This attribute is used to get the canvas paint window associated with a given canvas view.

Return Type: Canvas_paint_window
Procs: get
Objects: Canvas_view
Usage:

```c
Canvas_view view;
Xv_Window paint_window;

paint_window = (Xv_Window)xv_get(view,
        CANVAS_VIEW_PAINT_WINDOW);
```

See Also: 5.6.3.1, CANVAS_PAINT_CANVAS_WINDOW, CANVAS_PAINT_VIEW_WINDOW

**CANVAS_VIEWABLE_RECT**

Gets the visible part of the specified paint window in the paint window’s coordinates. This attribute operates on a Canvas object and requires as an argument, a handle to one of the canvas’ paint windows. The Rect * returned should not be freed, as it points to static storage.

Return Type: Rect *
Argument: Canvas_paint_window
Procs: get
Objects: Canvas
Usage:

```c
Rect *rect;
Xv_Window canvas_pw = canvas_paint_window(canvas);
rect = (Rect *)xv_get(canvas,
        CANVAS_VIEWABLE_RECT, canvas_pw);
```

See Also: 5.3

**CANVAS_WIDTH**

Specifies the width of the canvas paint window in pixels.

Argument: int
Default: In general, the canvas inherits its parent’s width.
Procs: create, get, set
Objects: Canvas
See Also: 5.4.2, CANVAS_MIN_PAINT_WIDTH

**CANVAS_X_PAINT_WINDOW**

This attribute controls the parameters used in the CANVAS_REPAINT_PROC. Setting this attribute to TRUE will cause the CANVAS_REPAINT_PROC to be called back with the parameters canvas, paint_window, display, xid, and xrectlist; setting CANVAS_X_PAINT_WINDOW to FALSE will cause it to be called back with canvas, paint_window, and repaint_area.
**CMS_BACKGROUND_PIXEL**

Returns the background pixel (index 0) in a colormap segment.

Return Type: unsigned long
Default: None
Procs: get
Objects: Cms
See Also: 21.3.1

**CMS_COLOR_COUNT**

Used to specify the number of colors being set with CMS_COLORS or CMS_X_COLORS. This can be used in conjunction with CMS_INDEX to set a range of the colors in the cms.

Argument: unsigned int
Default: CMS_SIZE for a non-control cms, or CMS_SIZE - CMS_CONTROL_COLORS for a control cms.
Procs: create, set
Objects: Cms
Usage: To set the colors in the array colors to the range from 10 to 14 in a cms, you would do the following:

```c
XColor colors[5];

xv_set(cms,
      CMS_X_COLORS, colors,
      CMS_INDEX, 10,
      CMS_COLOR_COUNT, 5,
      NULL);
```

See Also: 21.2.0.1

**CMS_COLORS**

Specifies the colors to be loaded into the colormap segment. Colors are specified as an array of Xv_singlecolor.

Argument: Xv_singlecolor
Default: None
Procs: create, get, set
Objects: Cms
See Also: 21.2.1.2

**CMS_CONTROL_CMS**

Indicates whether this colormap segment is a control colormap segment for use by control objects (like panel items and scrollbars).

Argument: int
Default: FALSE
Procs: create, get
Objects: Cms
See Also: 21.5
CMS_FOREGROUND_PIXEL

Returns the foreground pixel (index CMS_SIZE-1) in a colormap segment.

Return Type: unsigned long
Default: None
Procs: get
Objects: Cms
See Also: 21.3.1

CMS_INDEX

Specifies the starting index to CMS_COLOR_COUNT entries into the colormap.

Argument: unsigned long
Default: 0 (If control cms, CMS_CONTROL_COLORS)
Procs: create, set
Objects: Cms
See Also: 21.2.0.1, CMS_COLOR_COUNT

CMS_INDEX_TABLE

Used to translate the logical indices of the window’s colormap segment into actual pixel values.

Return Type: unsigned long *
Default: N/A
Procs: get
Objects: Cms
See Also: 21.3.0.1

CMS_NAME

Specifies the name of the colormap segment.

Argument: char *
Default: Unique name generated internally
Procs: create, get, set
Objects: Cms
See Also: 21.6

CMS_NAMED_COLORS

Specifies the names of the colors to be loaded into the colormap segment. The NULL-terminated list of color names is parsed by XParseColor().

Argument: List of char *
Default: None
Procs: create, set
Objects: Cms
See Also: 21.2.1.1

CMS_PIXEL

Translates a logical index into the actual colormap pixel value.

Argument: unsigned long
Default: None
Procs: get
Objects: Cms
Usage:

```c
unsigned long pixel;
pixel=(unsigned long)xv_get(cms, CMS_PIXEL, 2);
```

See Also: 21.3.0.1
CMS_SCREEN

Returns the screen with which the colormap segment is associated.

Return Type: Xv_Screen
Default: Default screen
Procs: get
Objects: Cms

CMS_SIZE

Specifies the size of the colormap segment.

Argument: int
Default: XV_DEFAULT_CMS_SIZE or CMS_CONTROL_COLORS, for a control cms
Procs: create, get
Objects: Cms
See Also: 21.2.0.1

CMS_TYPE

Specifies the type of the colormap segment. If set to XV_STATIC_CMS, then only read-only colors can be allocated from the cms. If set to XV_DYNAMIC_CMS, then only read-write colors can be allocated from the cms. XV_DYNAMIC_CMS can only be used with a dynamic visual.

Argument: Cms_type
Default: XV_STATIC_CMS
Procs: create
Objects: Cms
See Also: 21.1.1

CMS_X_COLORS

Specifies the colors to be loaded into the colormap segment. Colors are specified as an array of XColor.

Argument: XColor *
Default: None
Procs: create, set, get
Objects: Cms
See Also: 21.2.1.2

CURSOR_BACKGROUND_COLOR

Specifies the background color of a cursor as an RGB triplet.

Argument: Xv_singlecolor *
Default: white (255, 255, 255)
Procs: create, get, set
Objects: Xv_Cursor
See Also: 13.4

CURSOR_DRAG_STATE

Indicates whether the cursor is over a neutral zone (CURSOR_NEUTRAL), a valid drop zone (CURSOR_ACCEPT), or an invalid drop zone (CURSOR_REJECT). The shape of the cursor varies depending on the state. Note that the current drag and drop protocol does not support a “reject” cursor.

Argument: Cursor_drag_state
Default: CURSOR_NEUTRAL
Proc: create, get
Objects: Cursor
See Also: 13.5, CURSOR_DRAG_TYPE, CURSOR_STRING
CURSOR_DRAG_TYPE
Indicates whether the cursor is "move" (CURSOR_MOVE) or "copy" (CURSOR_DUPLICATE). The duplicate version has a shadow. When combined with CURSOR_STRING, you get either a text move or text duplicate cursor.

Argument: Cursor_drag_type
Default: CURSOR_MOVE
Procs: create, get
Objects: Cursor
See Also: 13.5, CURSOR_DRAG_STATE, CURSOR_STRING

CURSOR_FOREGROUND_COLOR
Specifies the foreground color of a cursor as an RGB triplet.

Argument: Xv_singlecolor *
Default: black (0, 0, 0)
Procs: create, get, set
Objects: Xv_Cursor
See Also: 13.4

CURSOR_IMAGE
Specifies the cursor's image.

Argument: Server_image
Default: None
Procs: create, get, set
Objects: Xv_Cursor
See Also: 13.1

CURSOR_MASK_CHAR
Specifies the index into the mask shape font. Predefined shapes are defined in <xview/cursor.h>.

Argument: unsigned int
Default: None
Procs: create, get, set
Objects: Xv_Cursor
Usage:

```
xv_set(cursor,
       CURSOR_MASK_CHAR, OLC_BASIC_MASK_PTR
       NULL);
```

See Also: 13.2, CURSOR_SRC_CHAR

CURSOR_OP
The value for this attribute is the rasterop (defined in <pixrect.h>) which will be used to paint the cursor.

Argument: int
Default: [ PIX_SRC | PIX_DST ]
Procs: create, get, set
Objects: Xv_Cursor
CURSOR_SRC_CHAR
Specifies the index into the shape font. Predefined shapes are in <xview/cursor.h>.
Argument: unsigned int
Default: -1
Procs: create, get, set
Objects: Xv_Cursor
Usage:

```c
xv_set(cursor,
    CURSOR_SRC_CHAR, OLC_BASIC_PTR,
    CURSOR_MASK_CHAR, OLC_BASIC_MASK_PTR,
    NULL);
```
See Also: 13.2, CURSOR_MASK_CHAR

CURSOR_STRING
Creates a text drag and drop cursor. The value of the attribute is the string which is to be displayed inside the “flying punch card.” If the string exceeds 3 characters, then only the first 3 characters are displayed, and a “More arrow” is shown within the cursor. CURSOR_STRING is mutually exclusive of CURSOR_IMAGE, CURSOR_SRC_CHAR, and CURSOR_MASK_CHAR. The string is not copied. Once the drag and drop operation is complete, the text cursor objects used in the operation must be destroyed.
Argument: char *
Default: None
Procs: create, get
Objects: Cursor
See Also: 13.5, CURSOR_DRAG_STATE, CURSOR_DRAG_TYPE

CURSOR_XHOT
Specifies the x coordinate of the hotspot. Its value cannot be negative.
Argument: int
Default: 0
Procs: create, get, set
Objects: Cursor
See Also: 13.3

CURSOR_YHOT
Specifies the y coordinate of the hotspot. Its value cannot be negative.
Argument: int
Default: 0
Procs: create, get, set
Objects: Cursor
See Also: 13.3

DND_ACCEPT_CURSOR
During a drag and drop operation, this defines the mouse cursor that is used when the cursor is over an acceptable drop-site.
Argument: Xv_cursor
Default: Predefined OPEN LOOK drag and drop cursor
Procs: create, get, set
Objects: Dnd
See Also: 19.3.3
**DND_ACCEPT_X_CURSOR**

An alternative to `DND_ACCEPT_CURSOR`, this attribute accepts an XID of a cursor instead of an `Xv_cursor`.

Argument: XID
Default: Predefined OPEN LOOK drag and drop cursor
Procs: create, get, set
Objects: Dnd
See Also: 19.3.3

**DND_CURSOR**

Defines the mouse cursor that will be used during the drag portion of the drag and drop operation.

Argument: Xv_cursor
Default: Predefined OPEN LOOK drag and drop cursor
Procs: create, get, set
Objects: Dnd
See Also: 19.3.3, DND_ACCEPT_CURSOR

**DND_TIMEOUT_VALUE**

Defines the amount of time to wait for an acknowledgment from the drop destination after the kicker message has been sent (`ACTION_DRAG_COPY` or `ACTION_DRAG_MOVE`). The kicker message is sent when the user releases the mouse button, forcing the drop.

Argument: struct timeval *
Default: Default selection package timeout
Procs: create, get, set
Objects: Dnd
See Also: 19.3.4

**DND_TYPE**

Defines whether this drag and drop operation will be a copy or a move. This is just a hint to the destination. If the type is a move operation and if the destination honors the hint, the destination will ask the source to convert the DELETE target.

Valid Values: DND_COPY or DND_MOVE
Default: DND_MOVE
Procs: create, get, set
See Also: 19.3

**DND_X_CURSOR**

An alternative to `DND_CURSOR`, but accepts an XID of a cursor instead of an `Xv_cursor`.

Argument: XID
Default: Predefined OPEN LOOK drag and drop cursor
Procs: create, get, set
Objects: Dnd
See Also: 19.3.3, DND_ACCEPT_X_CURSOR
**DROP_SITE_DEFAULT**
Specifies that this drop-site wants forwarded drops from the window manager. Such drops include drops on icons and window manager decorations. Only one drop-site default should be specified per base frame (specifying more than one will have unpredictable results). This attribute is only a hint to the window manager.

**Argument:** Bool  
**Default:** False  
**Procs:** create, get, set  
**Objects:** Drop_site_item  
**See Also:** 19.2

**DROP_SITE_DELETE_REGION**
Used to remove a region from the drop item. If the owner is destroyed with xv_destroy(), any drop-site regions attached to it will also be destroyed. When a NULL is passed as an argument, all regions in the drop-site will be removed.

**Argument:** Rect *  
**Default:** N/A  
**Procs:** create, set  
**Objects:** Drop_site_item  
**See Also:** 19.2.1

**DROP_SITE_DELETE_REGION_PTR**
Used to remove a list of regions from the drop item. Passing a NULL as an argument removes all regions in the drop-site.

**Argument:** Rect *(a NULL-terminated array of Rect structs)  
**Default:** N/A  
**Procs:** create, set  
**Objects:** Drop_site_item  
**See Also:** 19.2.1

**DROP_SITE_EVENT_MASK**
A mask used to specify if the regions within the site will receive synthetic previewing events. The previewing events will come with event_action() set to ACTION_DRAG_PREVIEW and event_id() set to one of LOC_WINENTER, LOC_WINEXIT or LOC_DRAG. These events will be delivered to the event procedure of the owner of the drop-site item. This mask is only a hint to the source. There is no guarantee the source will send these previewing events.

**Argument:** int  
**Valid Values:** [DND_ENTERLEAVE, DND_MOTION] Defined in <xview/dragdrop.h>  
**Default:** NULL  
**Procs:** create, get, set  
**Objects:** Drop_site_item  
**See Also:** 19.2.2.1

**DROP_SITE_ID**
An uninterpreted ID used to distinguish one drop-site from the next. Ideal when more than one site has been set on an object.

**Argument:** long  
**Default:** A value generated from xv_unique_key()  
**Procs:** create, get, set  
**Objects:** Drop_site_item  
**See Also:** 19.2.2.1
**DROP_SITE_REGION**

This attribute is used to associate a region to a drop-site item. The region is a `Rect *`. `DROP_SITE_REGION` will add to any existing regions within the drop-site item. The rect pointed to by `Rect *` will be copied. The coordinates in the rect should be relative to the drop-site item’s owner’s window. An `xv_get()` of a region of type `Rect *` will return an allocated `Rect` structure. This should be freed using `xv_free()` once the application has finished using it.

**Argument:** `Rect *`

**Default:** `NULL`

**Procs:** `create`, `get`, `set`

**Objects:** `Drop_site_item`

**See Also:** 19.2.1, `DROP_SITE_DELETE_REGION`

**DROP_SITE_REGION_PTR**

This attribute is similar to `DROP_SITE_REGION` except that it accepts a `NULL`-terminated array of regions. It will add to any existing regions that exist within the drop item. A `NULL` rect is defined to be one with width or height equal to 0. An `xv_get()` of a region list will return a `NULL`-terminated list of `Rects`. This data should be freed once the application has finished using it.

**Argument:** `Rect *` *(A `NULL`-terminated array of `Rect` structs)*

**Default:** `NULL`

**Procs:** `create`, `get`, `set`

**Objects:** `Drop_site_item`

**See Also:** 19.2.1, `DROP_SITE_DELETE_REGION`

**FILE_CHOOSER_ABBREV_VIEW**

Specifies if the `FILE_CHOOSER` should show invalid file names grayed out or not display them at all. An invalid file name is specified by the `FILE_CHOOSER_FILTER_FUNC` returning a value of `FILE_CHOOSER_IGNORE` or with an op set to `FILE_CHOOSER_NOT_MATCHED` for files, but not for directories. This only works for `FILE_CHOOSER_FILTER_STRING`. Note that the Save and Save As dialogs require that all documents are grayed out; setting this attribute on them will result in a display with nothing but folders and the “Go up one” entry.

**Argument:** `Bool`

**Default:** `FALSE`

**Procs:** `create`, `get`, `set`

**Objects:** `File_chooser`

**See Also:** Appendix D, *Version 3.2 and the File Chooser*, of *XView Programming Manual*.

**FILE_CHOOSER_APP_DIR**

Adds an application-specific path name to the Go To History Menu’s fixed space. The first argument is the string that will show on the `Menu_item`, and the second is the path name represented by this label. Both strings may be the same. The Application File Choosing Specification imposes a limit of five application-specific entries in the Go To menu. Any more than five unique entries will be silently truncated by the `FILE_CHOOSER` package.

Note that the Goto List is, by default, shared amongst the various `FILE_CHOOSER` instances, so setting this on one `FILE_CHOOSER` will affect the rest of the instances. Also, the Default History List used by `FILE_CHOOSER` ignores duplicate entries in the Recent Space.

**Argument 1:** `char *`

**Argument 2:** `char *`

**Procs:** `create`, `set`

**Objects:** `File_chooser`

**See Also:** Appendix D, *Version 3.2 and the File Chooser*, of *XView Programming Manual*.
FILE_CHOOSER_AUTO_UPDATE
Tells the FILE_CHOOSER not to re-read its current directory until either FILE_CHOOSER_UPDATE is called explicitly, or until the user performs some action to read the directory.

Argument:  Bool
Default:  TRUE
Proc: create, get, set
Objects:  File_chooser

FILE_CHOOSER_CD_FUNC
Specifies a callback to be invoked with an op of FILE_CHOOSER_BEFORE_CD before the FILE_CHOOSER attempts to change to another directory.

Argument:  int (*cd_func) ()
Proc: create, get, set
Objects:  File_chooser
Callback:

```c
int
cd_func( chooser, path, stats, op )
    File_chooser chooser;
    char * path;
    struct stat * stats;
    File_chooser_op op;
```

The expected return values are XV_OK or XV_ERROR. If XV_ERROR is returned, the directory change is effectively vetoed. It is expected that the application provides the user with the appropriate feedback.

The callback is invoked again with an op of FILE_CHOOSER_AFTER_CD after the contents of the scrolling list have been loaded, but before it has been displayed to the user. The return value is ignored from this invocation.

FILE_CHOOSER_CHILD
Get a handle to the opaque handles to the UI objects.

Warning: relying on values that use xv_set or xv_get from these handles is not guaranteed to be compatible in future releases.

Argument:  File_chooser_child
Valid Values:

- FILE_CHOOSER_GOTO_MESSAGE_CHILD
- FILE_CHOOSER_GOTO_BUTTON_CHILD
- FILE_CHOOSER_GOTO_PATH_CHILD
- FILE_CHOOSER_HISTORY_MENU_CHILD
- FILE_CHOOSER_CURRENT_FOLDER_CHILD
- FILE_CHOOSER_SELECT_MESSAGE_CHILD
- FILE_CHOOSER_FILE_LIST_CHILD
- FILE_CHOOSER_DOCUMENT_NAME_CHILD
- FILE_CHOOSER_OPEN_BUTTON_CHILD
- FILE_CHOOSER_SAVE_BUTTON_CHILD
- FILE_CHOOSER_CANCEL_BUTTON_CHILD
- FILE_CHOOSER_CUSTOM_BUTTON_CHILD

Proc: get
Objects:  File_chooser
FILE_CHOOSER_COMPARE_FUNC

Function to compare two entries in the list of files. Setting the FILE_CHOOSER_FILTER_FUNC to NULL does not guarantee any ordering to row entries.

Argument: int (*compare_func) ()
Default: Case-insensitive ascending sorting order. Also provided are case-insensitive descending and case sensitive ascending and descending.

Procs: create, get, set

Objects: File_chooser

Callback:

int
compare_func( row1, row2 )
File_chooser_row *row1;
File_chooser_row *row2;

where File_chooser_row is defined as:

typedef struct {
  char * file;
  struct stat * stats;
  File_chooser_op matched;
  char * xfrm;
} File_chooser_row;

file is a string representing the name of the file being sorted.
stats is a pointer to a stat structure as returned by the stat() system call.
matched is one of the values FILE_CHOOSER_MATCHED or FILE_CHOOSER_NOT_MATCHED, indicating if the file name has matched the regular expression installed using FILE_CHOOSER_FILTER_STRING.
xfrm field represents the file name, as returned by the strxfmt(3) function.

FILE_CHOOSER_CUSTOMIZE_OPEN

Allows the client to re-use the Open dialog within other contexts. Active only on a dialog of type FILE_CHOOSER_OPEN. The recommended uses are Insert, Include or Import.

The first argument is a string that will be used as the label for a special-purpose button at the bottom of the dialog. The Open button becomes Open Folder, as with a Save or Save As dialog. The second argument becomes the help message that is displayed immediately above the Scrolling List. The third argument is of type File_chooser_op indicating if this new type should be able to select files, or both files and directories (values are FILE_CHOOSER_SELECT_FILES or FILE_CHOOSER_SELECT_ALL).

The button added by this call will activate the FILE_CHOOSER_NOTIFY_FUNC, with the same arguments as the open callback and is retrievable using FILE_CHOOSER_CHILD with a value of FILE_CHOOSER_CUSTOM_BUTTON.

Argument 1: char *
Argument 2: char *
Argument 3: File_chooser_op

Procs: create

Objects: File_chooser

**FILE_CHOOSER_DIRECTORY**

Specifies the current working directory being displayed in the `FILE_CHOOSER`. This can be NULL.

Argument: char *
Default: The current working directory.
Procs: create, get, set
Objects: File_chooser

**FILE_CHOOSER_DOC_NAME**

Specifies the name of the application’s current document for use by the Save As dialog (this is only effective if the `FILE_CHOOSER_TYPE` attribute is `FILE_CHOOSER_SAVEAS`). It is suggested that this value be set to `current.doc.name` whenever a new document is created or opened.

Specifies the name of the default document name in a Save dialog (that is, this is only effective if the `FILE_CHOOSER_TYPE` attribute is `FILE_CHOOSER_SAVE`).

Argument: char *
Default: Untitled1
Procs: create, get, set
Objects: File_chooser

**FILE_CHOOSER_EXTEN_FUNC**

Function that gets called during resize handling that allows the client to adjust its extension controls for the new size of the dialog. The return value is the new height desired for the `exten_rect`, or -1 if there is no change. Note that new y values should be relative to the `exten_rect` and should expect to increase it upwards, because adding height to the extension area takes away from the number of rows in the Scrolling List. Newly defined height values should never exceed the `max_height` parameter. Clients may adjust the `FRAME_MIN_SIZE` attribute to prevent this from being a problem.

Returning a new height from this function is not the same as `xv_set` for the `FILE_CHOOSER_EXTENSION_HEIGHT` attribute.

Argument: int (*exten_func) ()
Procs: create, get, set
Objects: File_chooser

Callback:

```c
int exten_func(fc, frame_rect, exten_rect, left_edge, right_edge, max_height)
    File_chooser fc;
    Rect * frame_rect;
    Rect * exten_rect;
    int left_edge;
    int right_edge;
    int max_height;

    fc is the File Chooser instance.
    frame_rect is the new size of the Command Frame.
    exten_rect is the current size of the area delegated for extension items.
    left_edge is the position being used as the leftmost x position used by the regular objects on the File Chooser.
```
right_edge is the position being used as the rightmost point for regular item layout.
max_height is the maximum amount of space that the extension rect can occupy given the current size of the Frame.

It is recommended that the client set the PANEL_PAINT attribute to the value of PANEL_NONE for each xv_set call made on a Panel Item within this callback.

FILE_CHOOSER_EXTEN_HEIGHT
Tells the File Chooser to reserve this much vertical space for the added controls. Positioning controls outside this area is not supported by the XView File Chooser.

Argument: int
Procs: get, set
Objects: File_chooser

FILE_CHOOSER_FILTER_FUNC
Function called to validate the file names before they will be displayed to the user. The files for which this callback is invoked are chosen using FILE_CHOOSER_FILTER_MASK.

Argument: int (*filter_func) ()
Procs: create, get, set
Objects: File_chooser
Callback:

File_chooser_op
filter_func( fc, path, stats, matched, glyph,
            client_data, mask_glyph )
    File_chooser fc;
    char * path;
    struct stat * stats;
    File_chooser_op matched;
    Server_image * glyph;
    Xv_opaque * client_data;
    Server_image * mask_glyph;

fc is the File Chooser instance for which the callback was invoked.
path is the path name of the file being decided on.
statbuf is the address of a stat structure (see the stat(2) man page for details).
matched is one of the values FILE_CHOOSER_MATCHED or FILE_CHOOSER_NOT_MATCHED, depending on if the file name was found to match the regular expression given using the FILE_CHOOSER_FILTER_STRING attribute.
glyph and client_data are returnable values. If the filter function returns a Server_image in the glyph field, the glyph returned will be displayed in the list along with the file name. The Server_image returned is expected to be 16×16 pixels, regardless of the scale or font size of the File Chooser.
mask_glyph is the clip mask to be used with the glyph field. The Server_image returned must be of depth 1. In most cases, if the glyph is of depth 1, the mask_glyph is the same Server_image as the glyph.

If a value is returned in the client_data field, this same value will be returned, uninterpreted, to the callback installed from the FILE_CHOOSER_NOTIFY_FUNC attribute.

The expected return value from the filter-func is one of the ops FILE_CHOOSER_ACCEPT or FILE_CHOOSER_IGNORE. If the return value is FILE_CHOOSER_IGNORE, this file will
be displayed grayed out, and unselectable. This is the default behavior specified by the
Application File Choosing Specification. If the FILE_CHOOSER_ABBREV_VIEW attribute
is set to TRUE, these files will not be displayed at all.
Also, in a Save or Save As dialog, all documents will be set inactive automatically.

FILE_CHOOSER_FILTER_MASK
Mask bits for the FILE_CHOOSER_FILTER_FUNC.
Argument: int (File_chooser_filter_mask)
Valid Values: Defined by a set or an or of flags from the set of:
  FC_NONE_MASK
  FC_MATCHED_FILES_MASK
  FC_NOT_MATCHED_FILES_MASK
  FC_MATCHED_DIRS_MASK
  FC_NOT_MATCHED_DIRS_MASK
  FC_DOTDOT_MASK
  FC_ALL_MASK
Default: FC_MATCHED_FILES
Procs: create, get, set
Objects: File_chooser

FILE_CHOOSER_FILTER_STRING
Sets or gets an ex-like regular expression string which files are filtered through before being actively
displayed in the list. If no filter string is specified, all entries are assumed to match. “..” is always as-
sumed to match. A default glyph may be specified for entries that match the regular expression using
FILE_CHOOSER_MATCH_GLYPH.

Note that the Save and Save As dialogs must have all of the documents grayed out, so setting a filter
string on them will not be useful unless you are using a filter function to assign glyphs as well, even
though they are grayed out, or issuing FILE_CHOOSER_ABBREV_VIEW.
Argument: char *
Procs: create, get, set
Objects: File_chooser

FILE_CHOOSER_HISTORY_LIST
Specifies an opaque handle to the HISTORY_LIST object associated with the dialog instance. By de-
default, the FILE_CHOOSER package will look for a list identified by the string “XView GoTo History”
and create it if it cannot be found.
Argument: History_list
Procs: create, get, set
Objects: File_chooser
FILE_CHOOSER_MATCH_GLYPH

Sets or gets a Server_image to be used as the default glyph for all file names matching the filter string (see FILE_CHOOSER_FILTER_STRING attribute).

Argument: Server_image
Proc: create, get, set
Objects: File_chooser

FILE_CHOOSER_MATCH_GLYPH_MASK

Sets or gets Server_image to be used as a clip mask for the corresponding FILE_CHOOSER_MATCH_GLYPH. The Server_image must be of depth 1.

Argument: Server_image
Default: XV_NULL
Proc: create, get, set
Objects: File_chooser

FILE_CHOOSER_NO_CONFIRM

Tells the FILE_CHOOSER not to do confirmation before issuing the call to open or save callbacks.

Argument: Bool
Default: FALSE
Proc: create, get, set
Objects: File_chooser

FILE_CHOOSER_NOTIFY_FUNC

Callbacks invoked when the user selects a file from the dialog for opening or saving. The callback takes different form depending on whether the FILE_CHOOSER_TYPE is FILE_CHOOSER_OPEN or FILE_CHOOSER_SAVE/AS. Expected return values are XV_OK or XV_ERROR.

Argument: int (*notify_callback) ()
Proc: create, get, set
Objects: File_chooser
Callback 1:

```
int
open_callback( fc, path, file, client_data )
    File_chooser fc;
    char * path;
    char * file;
    Xv_opaque client_data;

    client_data is the client_data field set for this row in the list from the FILE_CHOOSER_FILTER_FUNC.
    path is the entire path to the file.
    The file string is strictly the file-name portion of the path. The directory portion of the path may be obtained with the FILE_CHOOSER_DIRECTORY attribute.
```
Callback 2:

```c
int
save_callback( fc, path, stats )
    File_chooser fc;
    char * path;
    struct stat * stats;
```

`path` is the full path of the file to be saved.
`stats` is a pointer to the file’s stat structure if the file exists, or `NULL` if the file does not exist.

If the file exists, it is up to the application to check the permissions from the stat structure and act accordingly. The File Chooser package will confirm with the user before overwriting an existing file prior to issuing the callback; this may be prevented with `FILE_CHOOSER_NO_CONFIRM`. Thus, if this callback is issued, the user expects the file to be written or overwritten. It is the responsibility of the client to alert the user of any failures with the appropriate feedback.

**FILE_CHOOSER_SAVE_TO_DIR**

If set `TRUE`, the Save/As: typein becomes blank and inactive, while the Save button remains active. In the default state (`FALSE`), the behavior is that the Save button is inactive whenever the Save/As: typein becomes empty.

Argument:  `Bool`
Default:  `FALSE`
Procs:  `create, get, set`
Objects:  `File_chooser`

**FILE_CHOOSER_SHOW_DOT_FILES**

Specifies whether or not the `FILE_CHOOSER` package should consider files beginning with the “.” prefix.

Argument:  `Bool`
Default:  `FALSE`
Procs:  `create, get, set`
Objects:  `File_chooser`

**FILE_CHOOSER_TYPE**

Specifies the type of dialog to create.

Argument:  `One of: FILE_CHOOSER_OPEN, FILE_CHOOSER_SAVE FILE_CHOOSER_SAVEAS`.
Procs:  `create, get`
Objects:  `File_chooser`

**FILE_CHOOSER_UPDATE**

Tells the `FILE_CHOOSER` to re-read its current directory and update its display.

Argument:  `None`
Procs:  `set`
Objects:  `File_chooser`

---

*XView Attributes* 51
FILE_LIST_ABBREV_VIEW
Tells the FILE_LIST package to “compress” out the files in the list for which the client returned FILE_LIST_IGNORE from the filter function (see FILE_LIST_FILTER_FUNC attribute). This behavior of graying out filenames, rather than leaving them out entirely, is specified in the Open Look File Choosing Spec as the default for a File Chooser object.

Argument:     Bool
Default:      FALSE
Procs:        create, get, set
Objects:      File_list

FILE_LIST_AUTO_UPDATE
Specifies whether the FILE_LIST package should automatically update the current list when the client changes an attribute that would modify the display (e.g., FILE_LIST_SHOW_DOT_FILES).

Argument:     Bool
Default:      TRUE.
Procs:        create, get, set
Objects:      File_list

FILE_LIST_CHANGE_DIR_FUNC
Sets or gets a function pointer that the client installs for the purpose of being notified for, and possibly to veto a change of directory. This can happen if the application sets the FILE_LIST_DIRECTORY attribute, or if the user double-clicks on a valid directory entry in the list (including the “go up one folder” entry.)

Argument:     int (*cd_func) ()
Procs:        create, get, set
Objects:      File_list
Callback:

    int
    cd_func ( item, path, statbuf, op )
    File_list item;
    char * path;
    struct stat * statbuf;
    File_list_op op;

The valid File_list_op values are FILE_LIST_BEFORE_CD and FILE_LIST_AFTER_CD.

The function is a notification of a pending change of directory to the new directory path. The callback is called both before and after the list is updated with the new directory. If the callback returns XV_ERROR, and the op is FILE_LIST_BEFORE_CD, than the pending change will be canceled, and the application will be assumed to provide the appropriate feedback to the user.

The return value is expected to be XV_OK or XV_ERROR.

The second time this function is called, the op value is equal to FILE_LIST_AFTER_CD. This informs the function that the call is after the list has been created, but before the list is displayed for the user. This gives you the opportunity to modify the list before displaying it.
FILE_LIST_COMPARE_FUNC

Used with the qsort library routine. Default function is case-insensitive ascending. Also provided
are case-insensitive descending and case-sensitive ascending and descending. See the functions and
macros section for a list of the provided comparison functions.

The comparison function is specified as follows:

Argument:  int (*compar_func)()
Proc:  create.get.set
Objects:  File_list
Callback:  int
  compar_func( row1, row2 )
  File_list_row *row1;
  File_list_row *row2;

The File_list_row structure includes the Panel_list_row_values struct for
this row, as well as the stat buffer and the matched flag. This will allow you to sort by
fields in any of these structures, if desired.

FILE_LIST_DIRECTORY

Sets or gets the directory currently being displayed in the list. If FILE_LIST_DIRECTORY is set to
NULL, the FILE_LIST will essentially become empty.

Argument:  char *
Default:  the current working directory
Proc:  create.get.set
Objects:  File_list

FILE_LIST_DOTDOT_STRING

Allows the client to modify the string used by the FILE_LIST package do denote the “..” entry.

Argument:  char *
Default:  “...Go up one folder...”
Proc:  create.get.set
Objects:  File_list

FILE_LIST_FILTER_STRING

Sets or gets an ex-like regular expression string which files are filtered through before being actively
displayed in the list. If no filter string is specified, all files are assumed to match. “..” is always assumed
to match.

If a file does not match, it is grayed out by default. This is not true of directories. Directories may
only be inactivated using FILE_LIST_FILTER_FUNC.

Argument:  char *
Default:  A default glyph may be specified for files that match the regular expression using
FILE_LIST_MATCH_GLYPH.
Proc:  create.get.set
Objects:  File_list
**file_list_filter_func**

Sets or gets a function pointer that the client installs for the purpose of okaying a file to be actively displayed in the list.

**Argument:**
int (*client_filter) ()

**Procs:**
create, get, set

**Objects:**
File_list

**Callback:**

```c
File_list_op
client_filter ( path, row )
    char * path;
    File_list_row * row;
```

path is a string that contains the absolute path of the file in question (except "..").

row is a File_list_row structure. File_list_row is defined as:

```c
typedef struct {
    File_list file_list;
    Panel_list_row_values vals;
    struct stat statbuf;
    File_list_op matched;
    char * xfrm;
} File_list_row;
```

file_list is the object handle of the File_list instance.
vals, the Panel_list_row_values, is as specified in <xview/panel.h>. These are the available parameters to a specific row in the PANEL_LIST. It can be conditionally modified from this callback; the exception is that the FILE_LIST package may modify the inactive field according to the return value from this function and the FILELISTABBREV_VIEW attribute. In addition, the extension_data field is reserved for use by the FILE_LIST package.

statbuf is the address of the current stat structure for the file.
matched is an indicator of whether or not the file name matched the regular expression and is one of FILE_LIST_MATCHED or FILE_LIST_NOT_MATCHED.
xfrm field contains the file name in the collated representation as returned by the strxfrm(3) function.

The return values for the callback are defined by File_list_op: FILE_LIST_IGNORE and FILE_LIST_ACCEPT. FILE_LIST_IGNORE means that the FILE_LIST package is not supposed to actively display this particular file name. FILE_LIST_ACCEPT tells the File_list package to actively display the entry.

Note: modifying the contents of the list from this callback it is not supported. To modify the list, use the attribute FILE_LIST_CHANGE_DIR_FUNC and wait for the FILE_LIST_AFTER_CD operation.
FILE_LIST_FILTER_MASK
Mask for FILE_LIST_FILTER_FUNC attribute.

Argument:  int (of type File_list_filter_mask)
Valid Values:

   FL_NONE_MASK
   FL_MATCHED_FILES_MASK
   FL_NOT_MATCHED_FILES_MASK
   FL_MATCHED_DIRS_MASK
   FL_NOT_MATCHED_DIRS_MASK
   FL_DOTDOT_MASK
   FL_ALL_MASK

Default:  FL_MATCHED_FILES
Procs:  create, get, set
Objects: File_list

FILE_LIST_MATCH_GLYPH
Sets or gets a glyph that is to be the default for all files that match the regular expression given by the
FILE_LIST_FILTER_STRING attribute. This does not apply to directory entries. Note: this is strictly
a convenience, this glyph may still be overridden with FILE_LIST_FILTER_FUNC.

Argument:  Server_image
Procs:  create, get, set
Objects: File_list

FILE_LIST_MATCH_GLYPH_MASK
Sets or gets a clip mask for the FILE_LIST_MATCH_GLYPH. The Server_image given must be of
depth 1.

Argument:  Server_image
Default:  XV_NULL
Procs:  create, get, set
Objects: File_list

FILE_LIST_ROW_TYPE
Takes a row number as argument. Returns a value of type File_list_row_type.

Argument:  int
Valid Values:

   FILE_LIST_DOTDOT_TYPE
   FILE_LIST_DIR_TYPE
   FILE_LIST_FILE_TYPE

Procs:  get
Objects: File_list
FILE_LIST_SHOW_DIR
Tells the FILE_LIST package to display its current directory path in the Title item of the list (uses the PANEL_LIST_TITLE attribute).

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: File_list

FILE_LIST_SHOW_DOT_FILES
Specifies whether or not the FILE_LIST package should consider files beginning with the "." prefix.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: File_list

FILE_LIST_UPDATE
Tells the FILE_LIST to re-read its current directory and update its display.

Argument: None
Procs: set
Objects: File_list

FILE_LIST_USE_FRAME
Tells the FILE_LIST package to display any error messages in the footer of the current Frame. If the current Frame has no footer, or the attribute is FALSE, the FILE_LIST package will use xv_error().

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: File_list

FONT_CHAR_HEIGHT
Returns the height (an int) of a specified character (a char) of the font. This is actually the height of the bounding rectangle, good for any character of the font.

Return Type: int
Argument: char
Procs: get
Objects: Xv_Font
Usage:

Xv_font font;
int height;
height = (int)xv_get(font, FONT_CHAR_HEIGHT, 'm');

See Also: 16.2
**FONT_CHAR_WIDTH**

Returns the width (int) of a specified character (char) of the font.

Return Type: int  
Argument: char  
Procs: get  
Objects: Xv_Font  
See Also: 16.2

**FONT_DEFAULT_CHAR_HEIGHT**

Returns the default character height of the font. Does not take a value.

Return Type: int  
Procs: get  
Objects: Xv_Font  
Usage:

```c
Xv_font font;
int height;
height = (int)xv_get(font, FONT_DEFAULT_CHAR_HEIGHT);
```

See Also: 16.2

**FONT_DEFAULT_CHAR_WIDTH**

Returns the default character width of the font. Does not take a value.

Return Type: int  
Procs: get  
Objects: Xv_Font  
See Also: 16.2

**FONT_FAMILY**

Specifies the name of a font family.

Argument: char *  
Default: None  
Procs: create, find, get  
Objects: Xv_Font  
See Also: 16.1.1

**FONT_INFO**

Returns a pointer to the X structure XFontStruct containing X-related information for the font.

Return Type: XFontStruct *  
Default: N/A  
Procs: get  
Objects: Xv_Font  
Usage:

```c
#include <X11/Xlib.h>
Xv_font font;
XFontStruct *font_info;
font_info = (XFontStruct *)xv_get(font, FONT_INFO);
```

See Also: 16.2
**FONT_NAME**

Specifies the name of the font desired. This takes precedence over all other Font attributes. The list of valid names that can be used can be displayed using the xlsfonts command.

Argument: char *
Procs: create, find
Objects: Xv_Font
Usage:

```c
Xv_font font;
font = xv_create (frame, FONT, 
    FONT_NAME, 
    "-adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1" 
    NULL); 
height = (int)xv_get(font, FONT_DEFAULT_CHAR_HEIGHT);
```

See Also: 16.1.4

**FONT_PIXFONT**

This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*. Returns the pixfont representation of the font.

**FONT_RESCALE_OF**

Given an existing font and a rescale factor, the returned font will be a similar font in the specified scale.

Argument1: Xv_Font
Argument2: Window_rescale_state (see <xview/window.h>)
Procs: create, find
Objects: Xv_Font
Usage:

```c
Xv_font font1, font2;
/* 
   * Find a font similar to font1 but in the large 
   * scale 
   */
    font2 = (Xv_FONT) xv_find (frame, FONT, 
        FONT_RESCALE_OF, font1, WIN_SCALE_LARGE, 
        NULL);
```

See Also: 16.1.3, FONT_SIZES_FOR_SCALE, FONT_SCALE

**FONT_SCALE**

Specifies the scale desired for a font. The scale settings map to certain pixel sizes (defaults are 10, 12, 14, and 19). If FONT_SIZE is used, it will take precedence over FONT_SCALE.

Argument: Window_rescale_state (see <xview/window.h>)
Default: WIN_SCALE_MEDIUM
Procs: create, find, get
Objects: Xv_Font
See Also: 16.1.3, FONT_RESCALE_OF, FONT_SIZES_FOR_SCALE
**FONT_SIZE**

Specifies the size of a font in pixels. Note that the valid values for size depend on what font sizes are available on the X server.

Argument: int
Default: 12
Procs: create, find, get
Objects: Xv_Font
Usage:

```c
Xv_font font;
/*
 * This creates a lucida font with normal style
 * with its size = 14
 */
font = (Xv_FONT) xv_find (frame, FONT
    FONT_FAMILY_LUCIDA,
    FONT_STYLE, FONT_STYLE_NORMAL
    FONT_SIZE, 14,
    NULL);
```

See Also: 16.1.2

**FONT_SIZES_FOR_SCALE**

Specifies a set of four integral sizes (measured in points) to which a font can be scaled.

Argument 1: int, for fonts scaled in small size
Argument 2: int, for fonts scaled in medium size
Argument 3: int, for fonts scaled in large size
Argument 4: int, for fonts scaled in extra_large size
Default: 10, 12, 14, 19
Procs: create, find
Objects: Xv_Font
Usage:

```c
Xv_font font;
/*
 * This creates a lucida font with normal style
 * with its size = 19
 */
font = (Xv_FONT) xv_find (frame, FONT,
    FONT_FAMILY_LUCIDA,
    FONT_STYLE, FONT_STYLE_NORMAL,
    FONT_SIZES_FOR_SCALE, 12, 14, 19, 26,
    FONT_SCALE, WIN_SCALE_LARGE,
    NULL);
```

See Also: 16.1.3, FONT_RESCALE_OF, FONT_SCALE
**FONT_STRING_DIMS**

Given a string and the address of a Font_string_dims structure (see `<xview/font.h>`), xv_get() fills it in with the width and height dimensions of the string. The pointer to the structure is returned by xv_get.

Return Type: Font_string_dims *
Argument1: char *
Argument2: Font_string_dims *
Procs: get
Objects: Xv_Font
Usage:

```c
Xv_font font;
Font_string_dims dims;

(void)xv_get (font, FONT_STRING_DIMS, "Hello World", dims);

/*
 * At this point 'dims' will contain the width and height
 * (in pixels) of the entire string "Hello World" as
 * rendered in the font 'font'.
 */
```

See Also: 16.2

**FONT_STYLE**

Specifies a font style.

Argument: char *
Default: FONT_STYLE_NORMAL
Procs: create, find, get
Objects: Xv_Font
See Also: 16.1.1

**FRAME_ACCELERATOR**

Specifies a window-level accelerator. The character in the first argument is used to call the procedure in the second argument with the data in the third argument. On xv_get, the frame package searches through the linked list of accelerators, and returns a pointer to the Frame_accelerator structure whose code or keysym matches the specified code and keysym. Applications do not normally use xv_get with this attribute. Note: Certain key combinations are reserved for semantic actions within XView and should not be used for accelerators. These key combinations are listed in Appendix C, Mouseless Model Keyboard Mappings, of the XView Programming Manual.

On create, set:

- Argument 1: char
- Argument 2: void (*)(())
- Argument 3: Xv_opaque

On get:

- Argument 1: char
- Argument 2: KeySym
- Procs: create, get, set
- Objects: Frame
Callback:

```c
void accelerator_notify_proc(value, event)
    Xv_opaque value; /* from Argument 3 */
    Event *event;
```

See Also: 6.14

**FRAME_BACKGROUND_COLOR**

This attribute is obsolete. To change a frame’s background color, create a cms and set it on the frame.

**FRAME_BUSY**

Sets label to gray and changes cursor to hour-glass.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Frame
See Also: 4.5

**FRAME_CLOSED**

Controls the frame’s mapped state (either open or iconic).

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Frame
See Also: 14.2, XV_SHOW

**FRAME_CLOSED_RECT**

Sets the size of the frame’s icon. To comply with the normal window manager placement strategy, if only the size of the icon is being set and not its screen location, first get the current `FRAME_CLOSED_RECT` rect and use the retrieved `rect.r_top` and `rect.r_left` values in the call to set `FRAME_CLOSED_RECT`.

Argument: Rect *
Default: rect.r_width = 64
    rect.r_height = 64
    rect.r_left = window manager dependent value
    rect.r_top = window manager dependent value
Procs: create, get, set
Objects: Frame
See Also: 4.2.3

**FRAME_CMD_DEFAULT_PIN_STATE**

This attribute controls the initial state of the pin when the frame goes from unmapped (withdrawn) to mapped state. It is valid for both mapped and unmapped frames. However, if the frame is currently mapped, the effects of the change will be visible only on the next transition from unmapped to mapped state.

Argument: int
Valid Values: [FRAME_CMD_PIN_IN, FRAME_CMD_PIN_OUT] defined in `<xview/frame.h>`.
Procs: create, get, set
Objects: Frame_cmd
See Also: 4.3.2, FRAME_CMD_PIN_STATE
FRAME_CMD_PANEL

Gets the default panel in the command frame.

Return Type: Panel
Procs: get
Objects: Frame
See Also: 11.12

FRAME_CMD_PIN_STATE

This attribute returns the current state of the pin. It is valid for both mapped and unmapped frames.

Though, for unmapped frames it will always return FRAME_CMD_PIN_OUT.

Argument: int
Valid Values: [FRAME_CMD_PIN_IN, FRAME_CMD_PIN_OUT] defined in <xview/frame.h>
Procs: get, set
Objects: Frame_cmd
See Also: 4.3.2, FRAME_CMD_DEFAULT_PIN_STATE

FRAME_CMD_PUSHPIN_IN

This attribute is obsolete. It is supported only for compatibility reasons. Applications should use FRAME_CMD_DEFAULT_PIN_STATE and FRAME_CMD_PIN_STATE instead. Indicates whether the pushpin is in or out.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Frame
See Also: FRAME_CMD_PIN_STATE

FRAME_DEFAULT_DONE_PROC

The default procedure is to set the subframe to WIN_SHOW, FALSE.

Default: FRAME_DEFAULT_DONE_PROC
Procs: create, get, set
Objects: Frame
See Also: 4.3.3, FRAME_DONE_PROC

FRAME_DONE_PROC

Names a procedure to be called when the command frame is dismissed by the user (by taking the pushpin out).

Argument: void (*frame_done_proc)()
Default: FRAME_DEFAULT_DONE_PROC
Procs: create, get, set
Objects: Frame
Callback:

void

frame_done_proc(frame)
    Frame frame;

See Also: 4.3.3
FRAME_FOCUS_DIRECTION

The direction in which the Location Cursor is pointing.

Argument:  Frame_focus_direction(enum)
Valid Values:  {FRAME_FOCUS_RIGHT, FRAME_FOCUS_UP}
Procs:  create, set, get
Objects:  Frame
See Also:  6.13.4

FRAME_FOCUS_WIN

Returns the handle of the Location Cursor (focus) window.

Return Type:  Xv_window
Procs:  get
Objects:  Frame
See Also:  6.13.4

FRAME_FOREGROUND_COLOR

This attribute is obsolete. To change a frame’s foreground color, create a cms and set it on the frame.

FRAME_ICON

Identifies the base frame’s icon.

Argument:  Xv_opaque
Default:  A default empty icon.
Procs:  create, get, set
Objects:  Frame
See Also:  4.2.3

FRAME_INHERIT_COLORS

This attribute is obsolete. To implement this functionality, use WIN_INHERIT_COLORS on the frame.

FRAME_LABEL

Specifies the label used in the window manager’s titlebar for the frame. XView copies the string on set.

Argument:  char *
Default:  NULL
Procs:  create, get, set
Objects:  Frame
See Also:  4.2.2, FRAME_SHOW_LABEL

FRAME_LEFT_FOOTER

Specifies the left-justified footer. XView copies the string on set.

Argument:  char *
Default:  NULL
Procs:  create, get, set
Objects:  Frame
See Also:  FRAME_SHOW_FOOTER, FRAME_RIGHT_FOOTER
FRAME_MAX_SIZE

This attribute is similar to FRAME_MIN_SIZE, but allows the application programmer to specify a maximum size the frame can be resized to by a user. All other aspects of this attribute are exactly the same as FRAME_MIN_SIZE.

Return Type: void
Argument 1: int (maximum width of frame)
Argument 2: int (maximum height of frame)
Default: The default value is 0. In other words there is no application specified minimum or maximum size. Keep in mind that the window manager may impose one though.

Procs: create, get, set
Objects: Frame
See Also: 4.3.5

FRAME_MENU

For create and set, this replaces current menu list with the one passed on the avlist. For get, this returns the current list of menus. The list returned should not be modified by the application. The number of menus can be obtained with FRAME_MENU_COUNT.

Argument: Null terminated list of Menus. On get, this returns (Menu *).
Default: None
Procs: create, get, set
Objects: Frame
Usage:

```
        xv_set(frame1, FRAME_MENUS, edit_menu, load_menu,
              NULL, NULL);
```

FRAME_MENU_COUNT

Returns the current number of menus registered on the frame using FRAME_MENUS, FRAME_MENU_ADD, or FRAME_MENU_DELETE.

Argument: int
Default: None
Procs: get
Objects: Frame
Usage:

```
        int menu_count;
        menu_count = (int)xv_get(frame2, FRAME_MENU_COUNT);
```

FRAME_MENU_ADD

Appends to the list of accelerated menus on the frame.

Argument: Menu
Default: None
Procs: create, set
Objects: Frame
Usage:

```
        xv_set(frame1,
               FRAME_MENU_ADD, print_menu, NULL);
```
**FRAME_MENU_DELETE**

Deletes from the list of accelerated menus on the frame.

Argument: Menu
Default: None
Procs: create, set
Objects: Frame
Usage:

```c
xv_set(frame1, FRAME_MENU_DELETE, print_menu, NULL);
```

**FRAME_MIN_SIZE**

This attribute allows the application programmer to specify a minimum size a frame can be resized to by a user. The FRAME_MIN_SIZE attribute takes two integer parameters, specifying the minimum width and height of the WM_NORMAL_HINTS property. Note that the minimum size is only a hint to the window manager. Some window managers may choose to ignore certain application specified hints. Setting both the minimum width and height to 0 effectively removes any application controlled minimum restriction on size.

Return Type: void
Argument 1: int (minimum width of frame)
Argument 2: int (minimum height of frame)
Default: The default value is 0. In other words there is no application specified minimum or maximum size. Keep in mind that the window manager may impose one. Defaults to a pre-determined number of rows and columns for a FILE_CHOOSER. It is recommended that clients who extend the FILE_CHOOSER update the FRAME_MIN_SIZE accordingly. Note: the client may increase the min size of the Frame, but decreasing it below the default values is not supported. Should not be set at create time for a File Chooser.
Procs: create, get, set
Objects: Frame
Usage: To get the previous set values of FRAME_MIN_SIZE use xv_get() and pass in two parameters:

```c
int width, height;
(void)xv_get(frame, FRAME_MIN_SIZE, &width, &height);
```

Note that some window managers may choose to ignore changes to the WM_NORMAL_HINTS property on frames that are already mapped. Thus, depending on the window manager, it may be necessary to unmap and then remap the frame before the FRAME_MIN_SIZE values take effect.

See Also: 4.3.5

**FRAME_NEXT_PANE**

Set the input focus to the next pane that can accept input focus.

Argument: None
Valid Values: {FRAME_FOCUS_RIGHT, FRAME_FOCUS_UP}
Procs: set
Objects: Frame
FRAME_NO_CONFIRM
Controls whether a notice is displayed when a frame is destroyed.

Argument:  Bool
Default:  TRUE
Procs:  create, get, set
Objects:  Frame
See Also:  4.2.4

FRAME_NTH_SUBFRAME
Gets the frame’s nth (from 1) subframe. Returns NULL if requested Subframe does not exist.

Return Type:  Frame
Argument:  int
Procs:  get
Objects:  Frame
Usage:

Frame frame;
frame = xv_get (base_frame, FRAME_NTH_SUBFRAME, 1);

See Also:  4.8, FRAME_NTH_SUBWINDOW

FRAME_NTH_SUBWINDOW
Gets the frame’s nth (from 0) subwindow. Returns NULL if requested Subwindow does not exist.

Return Type:  Xv_Window
Argument:  int
Procs:  get
Objects:  Frame
See Also:  4.8, FRAME_NTH_SUBFRAME

FRAME_PREVIOUS_ELEMENT
When set on a canvas, the focus is set to the view’s horizontal scrollbar, vertical scrollbar, or the last element in the canvas.

Argument:  None
Procs:  set
Objects:  Frame

FRAME_PREVIOUS_PANE
Set the input focus to the previous pane that can accept input focus.

Argument:  None
Procs:  set
Objects:  Frame

FRAME_RIGHT FOOTER
Specifies the right-justified footer. XView copies the string on set.

Argument:  char *
Default:  NULL
Procs:  create, get, set
Objects:  Frame
See Also:  FRAME_SHOW FOOTER, FRAME_LEFT FOOTER
FRAME_SHOW_HEADER
Indicates whether the header is visible. This is only a hint to the window manager. Some window managers may not honor this hint, some window managers may only honor this hint when the frame leaves the withdrawn state. Thus, to see the effect of setting this attribute to FALSE, the frame may need to be unmapped, and then mapped again.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Frame
See Also: 4.2.2

FRAME_SHOW_LABEL
Indicates whether the frame’s label is displayed. This is equivalent to FRAME_SHOW_HEADER.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Frame

FRAME_SHOW_RESIZE_CORNER
Determines whether a frame has resize corners. This is only a hint to the window manager. Some window managers may not honor this hint, some window managers may only honor this hint when the frame leaves the withdrawn state. Thus, to see the effect of setting this attribute to FALSE, the frame may need to be unmapped, and then mapped again. This attribute has no effect on window managers that are not OPEN LOOK compliant.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Frame
See Also: 4.3.4

FRAME_WM_COMMAND_ARGC
Returns the number of command-line option strings stored on the frame.

Return Type: int
Default: 0
Procs: get
Objects: Frame
See Also: 4.11, FRAME_WM_COMMAND_ARGCARGV, FRAME_WM_COMMAND_ARGV, FRAME_WM_COMMAND_STRINGS, WIN_CMD_LINE, Chapter 12, Interclient Communication, in the Xlib Programming Manual.
FRAME_WM_COMMAND_ARGC_ARGV

Lets an application set the command-line options that can be used to (re)start it. The options passed, in addition to XView options, are stored on a property called WM_COMMAND on the frame window. The options passed are stored by XView and will be added to the XView options on the WM_COMMAND property on the frame window, upon receiving a WM_SAVE_YOURSELF request from the session/window manager. The program xprop can be used to display a window’s properties. Only one base frame window of the application needs to have this property set. This property is read possibly by a session manager to restart clients. Setting this attribute’s arguments to NULL and -1 prevents any command-line option information from being saved on the frame. If there are two or more base frames in the application, the second and subsequent base frames should set their FRAME_WM_COMMAND_ARGC_ARGV attributes’ arguments to NULL and -1 if they want to avoid multiple invocations of the same application by the session manager. The first argument is the number of strings passed in the second argument. The second argument is a pointer to an array containing the command-line option strings. The strings passed are copied and cached on the frame.

Argument1: int
Argument2: (char **)
Default: 0 for Argument1 NULL for Argument2
Procs: create, set
Objects: Frame
Usage:

Frame base_frame, second_frame;
char *argv[10];
int argc = 0;

argv[argc++] = "-I"
argv[argc++] = "ls"
argv[argc++] = "-bold_font"
argv[argc++] = "courier-bold-14"

/*
 * This ensures that the above options are stored
 * on the base frame
 */
xv_set(base_frame, FRAME_WM_COMMAND_ARGC_ARGV,
       argc, argv, NULL);

/*
 * This ensures that no command-line information will
 * be stored on this frame.
 */
xv_set(second_frame,
       FRAME_WM_COMMAND_ARGC_ARGV, NULL, -1, NULL);

See Also: 4.11, 20.9.8, FRAME_WM_COMMAND_ARGC_ARGV, FRAME_WM_COMMAND_ARGC,
FRAME_WM_COMMAND_STRINGS, WIN_CMD_LINE. Chapter 12, Interclient
FRAME_WM_COMMAND_ARGV

Returns the array containing the command-line option strings stored on the frame. The strings in the array must not be modified by client programs. If the value returned is -1, this means that no command-line information is stored on the frame.

Return Type: (char **) or -1
Default: NULL
Procs: get
Objects: Frame
See Also: 4.11, FRAME_WM_COMMAND_ARGC_ARGV, FRAME_WM_COMMAND_ARGC, FRAME_WM_COMMAND_STRINGS, WIN_CMD_LINE, Chapter 12, Interclient Communication, in the Xlib Programming Manual.

FRAME_WM_COMMAND_STRINGS

Lets an application set the command-line options that can be used to (re)start it. The options passed, in addition to XView options are stored on a property called WM_COMMAND on the frame window. The options passed are stored by XView and will be added to the XView options on the WM_COMMAND property on the frame window, upon receiving a WM_SAVE_YOURSELF request from the session/window manager. (The program xprop can be used to display a window’s properties.) Only one base frame window of the application needs to have this property set. This property is read, possibly by a session manager to restart clients. Setting this attribute to -1 prevents any command-line option information from being saved on the frame. If there are two or more base frames in the application, the second and subsequent base frames should set their FRAME_WM_COMMAND_ARGC_ARGV attributes to -1 if they want to avoid multiple invocations of the same application by the session manager. The strings passed are copied and cached on the frame.

Argument: NULL-terminated list of (char *) or -1 followed by NULL
Default: None
Procs: create, set
Objects: Frame
Usage:

```c
Frame base_frame, second_frame;

/* Ensure that the given options are stored on *
 * on the base frame *
*/
xv_set(base_frame, FRAME_WM_COMMAND_STRINGS,
   "-l",
   "ls",
   "-bold_font",
   "courier-bold-14",
   NULL,
   NULL);

/* This ensures that no command-line information will *
 * be stored on this frame. *
*/
xv_set(second_frame, FRAME_WM_COMMAND_STRINGS,
   -1, NULL, NULL);
```

See Also: 4.11, FRAME_WM_COMMAND_ARGC_ARGV, FRAME_WM_COMMAND_ARGC, FRAME_WM_COMMAND_ARGV, WIN_CMD_LINE, Chapter 12, Interclient Communication, in the Xlib Programming Manual.
FRAME_X_ACCELERATOR
Specifies a window-level accelerator. The keysym in the first argument is used to call the procedure in the second argument with the data in the third argument. On xv_get, the Frame package searches through the linked list of accelerators, and returns a pointer to the Frame_accelerator structure whose code or keysym matches the specified code and keysym. Applications do not normally use xv_get with this attribute. Note: Certain key combinations are reserved for semantic actions within XView and should not be used for accelerators. These key combinations are listed in Appendix C, Mouseless Model Keyboard Mappings of the XView Programming Manual.

Argument 1: KeySym
Argument 2: void (*accelerator_notify_proc)()
Argument 3: Xv_opaque
Procs: create, get, set
Objects: Frame
Callback:

```
void
accelerator_notify_proc(value, event)
Xv_opaque value; /* from Argument 3 */
Event *event;
```

See Also: 6.14

FULLSCREEN_ALLOW_EVENTS
When in a fullscreen grab and the pointer and/or the keyboard is “frozen,” specifies how the events that are queued up in the server due to X grabs, are processed.

Argument: int (where the value is any of the AllowEvents modes in <X11/X.h>)
Procs: create, set
Objects: Fullscreen
See Also: XAllowEvents(), Xlib Reference Manual.

FULLSCREEN_ALLOW_SYNC_EVENT
When in a synchronous grab mode, where the keyboard and/or pointer is frozen by a client; specifies that normal event processing continue until the next mouse button or keyboard event.

Argument: No value
Procs: create, set
Objects: Fullscreen

FULLSCREEN_CURSOR_WINDOW
Restricts the cursor to a specified window. xv_get identifies the window.

Argument: Xv_window
Default: None
Procs: create, get
Objects: Fullscreen

FULLSCREEN_GRAB_KEYBOARD
Specifies whether keyboard should be grabbed.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Fullscreen
FULLSCREEN_GRAB_POINTER
   Specifies whether pointer should be grabbed.
   Argument:   Bool
   Default:   TRUE
   Procs:   create, get, set
   Objects:   Fullscreen

FULLSCREEN_GRAB_SERVER
   Specifies whether server should be grabbed.
   Argument:   Bool
   Default:   TRUE
   Procs:   create, get, set
   Objects:   Fullscreen

FULLSCREEN_INPUT_WINDOW
   Specifies the window from which input is read; xv_get identifies that window. The server, keyboard, and pointer are grabbed for this window.
   Argument:   Xv_window
   Default:   Owner window
   Procs:   create, get
   Objects:   Fullscreen

FULLSCREEN_KEYBOARD_GRAB_KBD_MODE
   Determines the grab mode for the keyboard when grabbing the keyboard.
   Argument:   Fullscreen_grab_mode
   Default:   FULLSCREENASYNCHRONOUS
   Procs:   create, get, set
   Objects:   Fullscreen

FULLSCREEN_KEYBOARD_GRAB_PTR_MODE
   Determines the grab mode for the pointer when grabbing the keyboard.
   Argument:   Fullscreen_grab_mode
   Default:   FULLSCREENASYNCHRONOUS
   Procs:   create, get, set
   Objects:   Fullscreen

FULLSCREEN_OWNER_EVENTS
   When a window grabs the server/keyborad/pointer, the value of this attribute determines the distribution of events to the application’s windows.

   All events occurring outside all the application’s windows, are reported to the grab window (see FULLSCREEN_INPUT_WINDOW).

   For events occurring within the application’s windows: If FULLSCREEN_OWNER_EVENTS is TRUE, the events are reported to the window indicated by the pointer. If FULLSCREEN_OWNER_EVENTS is FALSE, the events are reported to the grab window.
   Argument:   Bool
   Default:   FALSE
   Procs:   create, get, set
   Objects:   Fullscreen

XView Attributes  71
FULLSCREEN_PAINT_WINDOW

Specifies or gets the paint window that will be or is already in fullscreen.

Argument: Xv_opaque
Default: Owner window
Procs: create, get
Objects: Fullscreen

FULLSCREEN_POINTER_GRAB_KBD_MODE

Determines the grab mode for the keyboard when grabbing the pointer.

Argument: Fullscreen_grab_mode
Default: FULLSCREENASYNCHRONOUS
Procs: create, get, set
Objects: Fullscreen

FULLSCREEN_POINTER_GRAB_PTR_MODE

Determines the grab mode for the pointer when grabbing the pointer.

Argument: Fullscreen_grab_mode
Default: FULLSCREENASYNCHRONOUS
Procs: create, get, set
Objects: Fullscreen

FULLSCREEN_RECT

Returns a pointer to the rectangle containing the paint window that is currently fullscreen.

Argument: Rect *
Default: Owner window's bounding box
Procs: get
Objects: Fullscreen

FULLSCREEN_SYNC

Specifies whether to grab in synchronous (TRUE) or asynchronous (FALSE) mode.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Fullscreen

HELP_STRING_FILENAME

The name of a file containing a list of string pairs. The file is searched for in the directories listed in the environment variable HELPPATH. Each line in the file contains two words: the first word is the help string for which help is available, and the second word is of the form file:target, which XView uses to find the Spot Help text and More Help data. The first word must be less than 128 characters, and the second less than 64 characters. HELP_STRING_FILENAME is to be set on the paint window, or any of its owners, where the strings are to be painted.

Argument: Char *
Default: No string-help available
Procs: create, get, set
Objects: Icon
See Also: 23.2.3
HISTORY_ADD_FIXED_ENTRY
Add a string to the Fixed Space in the list. A Fixed String is always added to the bottom of the Fixed Space in the list. Passing a label of NULL will add a blank Menu_item. This attribute takes a pair of strings, the first of which is the label displayed in the Menu, and the second is the value of the menu item associated with this entry.

Argument 1: char *
Argument 2: char *
Default: None
Procs: create, set
Objects: History_list

HISTORY_ADD_ROLLING_ENTRY
Add a string to the Rolling Space. Strings in the Rolling Space are stacked and will roll off after some specified number of strings are added. This attribute takes a pair of strings, the first of which is the label displayed in the Menu, and the second is the value of the menu item associated with this entry.

Argument 1: char *
Argument 2: char *
Default: None
Procs: create, set
Objects: History_list

HISTORY_DUPLICATE_LABELS
Allow duplicate labels in the Rolling Space. Checks both Fixed and Rolling Space for duplicates.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: History_list

HISTORY_DUPLICATE_VALUES
Allow duplicate values in the Rolling Space. Checks both Fixed and Rolling Space for duplicates.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: History_list

HISTORY_FIXED_COUNT
Returns the number of entries currently in the Fixed Space.

Argument: int
Default: None
Procs: get
Objects: History_list
**HISTORY_INACTIVE**
Sets/Gets the Inactive state for a specified row. The first argument is one of HISTORY_FIXED or HISTORY_ROLLING. The second argument is the row number, and the third argument is the state of the flag (set only). If the row does not exist, the xv_set call is ignored, or the xv_get call will return -1.

Argument 1: enum
Argument 2: int
Argument 3: int
Default: FALSE
Procs: get, set
Objects: History_list

**HISTORY_LABEL**
Returns the value or label from a specified space (defined HISTORY_FIXED or HISTORY_ROLLING), for a specified row number. If the specified row does not exist, NULL will be returned.

Argument 1: enum
Argument 2: int
Default: None
Procs: get
Objects: History_list

**HISTORY_MENU_HISTORY_LIST**
Specifies the HISTORY_LIST object that is associated with this HISTORY_MENU instance.

Argument: History_list
Default: NULL
Procs: create, get, set
Objects: History_menu

**HISTORY_MENU_OBJECT**
Retrieves the handle to the internal Menu. Note: the only supported use of this handle is to pass it to menu_show() or to another XView object (such as a PANEL_BUTTON) for its menu argument.

Argument: No value
Procs: get
Objects: History_menu

**HISTORY_NOTIFY_PROC**
The callback invoked by the user’s selection on the HISTORY_MENU.

Argument: int (*) ()
Default: NULL
Procs: create, get, set
Objects: History_menu
Callback:

```c
void
callback ( hm, label, value )

History_menu  hm;
char *  label;
char *  value;
```

**HISTORY_ROLLING_COUNT**

Returns the number of entries currently in the Rolling Space.

- **Argument:** int
- **Default:** None
- **Proc:** get
- **Objects:** History_list
- **See Also:** Appendix D, *Version 3.2 and the File Chooser*, of *XView Programming Manual*.

**HISTORY_ROLLING_MAXIMUM**

Specifies the maximum strings in the Rolling Space before roll off actually occurs.

- **Argument:** int
- **Default:** 15
- **Proc:** create, get, set
- **Objects:** History_list
- **See Also:** Appendix D, *Version 3.2 and the File Chooser*, of *XView Programming Manual*.

**HISTORY_VALUE**

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

- **Argument 1:** int
- **Argument 2:** int
- **Default:** None
- **Proc:** get
- **Objects:** History_list
- **See Also:** Appendix D, *Version 3.2 and the File Chooser*, of *XView Programming Manual*.

**ICON_FONT**

Specifies the icon’s font.

- **Argument:** Xv_font
- **Default:** lucida medium
- **Proc:** create, get, set
- **Objects:** Icon

**ICON_HEIGHT**

Icon’s height in pixels.

- **Argument:** int
- **Default:** 64
- **Proc:** create, get, set
- **Objects:** Icon
**ICON_IMAGE**
Sets or gets the remote image for icon’s image.

**Argument:** Server_image

**Default:** NULL

**Procs:** create, get, set

**Objects:** Icon

See Also: 14.2.0.1

**ICON_IMAGE_RECT**
Sets or gets the bounding box (rect) for the icon’s image.

**Argument:** Rect *

**Default:** Origin (0,0), width 64, height 64

**Procs:** create, get, set

**Objects:** Icon

See Also: 14.2

**ICON_LABEL**
Specifies the icon’s label.

**Argument:** char *

**Default:** NULL

**Procs:** create, get, set

**Objects:** Icon

See Also: 14.2.1

**ICON_LABEL_RECT**
Sets or gets the bounding box for the icon’s label. Relative to the icon, 0,0 is the upper-left corner of the icon.

**Argument:** Rect *

**Default:** Bottom left-hand corner of the icon

**Procs:** create, get, set

**Objects:** Icon

See Also: 14.2.1

**ICON_MASK_IMAGE**
The icon’s GC’s clipmask is set to this bitmap. If this is set, ICON_TRANSPARENT is set to TRUE as a side effect.

**Argument:** Pixmap or Server_image

**Default:** FALSE

**Procs:** create, get, set

**Objects:** Icon

See Also: 14.2.0.3

**ICON_TRANSPARENT**
Sets the background color of the icon to be the same as the workspace’s background color.

**Argument:** int

**Default:** FALSE

**Procs:** create, get, set

**Objects:** Icon

See Also: 14.2.0.2
ICON_TRANSPARENT_LABEL

Draws the given string into an icon using the foreground only. It does not affect any other pixels in the bounding box for each character. Creating, setting, and getting ICON_TRANSPARENT_LABEL is equivalent to creating, setting, and getting ICON_LABEL except that the string is drawn in the foreground color only.

Argument:  char *
Default:  NULL
Procs:  create, get, set
Objects:  Icon
See Also:  14.2.2

ICON_WIDTH

Icon's width in pixels.

Argument:  int
Default:  64
Procs:  create, get, set
Objects:  Icon

MENU_ACCELERATOR

When used in a create or set call, sets an accelerator on a menu item. If an accelerator is changed with set, FRAME_MENUS must be set again before the accelerator will take effect (see FRAME_MENUS).

The accelerator string will be copied by XView.

Get will return the accelerator string. The returned string should not be modified.

Argument:  (char *) The accelerator string. For example “Meta+f,” “Meta+Ctrl+L.”
Default:  None
Procs:  create, get, set
Objects:  Menu_item

```c
Menu menu;

menu = xv_create(NULL, MENU
    MENU_ITEM,
    MENU_STRING, "Load",
    MENU_NOTIFY_PROC, load_proc,
    MENU_ACCELERATOR, "Meta+l",
    NULL,
    NULL);

Menu_item load_item;

load_item = xv_create(NULL, MENU_ITEM,
                          MENU_STRING, "Load",
                          MENU_NOTIFY_PROC, load_proc,
                          MENU_ACCELERATOR, "Meta+l",
                          NULL);
```

MENU_ACTION_ACCELERATOR

Creates a menu item with a given label, notify procedure and accelerator. If an accelerator is changed with set, FRAME_MENUS must be set again before the accelerator will take effect (see FRAME_MENUS).

The menu item label string, argument 1, will not be copied by XView. The accelerator string, argument 3, will be copied by XView.
Argument 1: (char *) Menu item label
Argument 2: void (*)() Notify procedure to call for this accelerator.
Argument 3: (char *) The accelerator string for example, “Meta+f,” “Meta+Ctrl+L.”
Default: None
Procs: create, set
Objects: Menu
Usage:

```c
xv_set(menu,
       MENU_ACTION_ACCELERATOR,
       "Load", load_proc, "Meta+L",
       NULL);
```

**MENU_ACTION_IMAGE**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_ACTION_ITEM**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_APPEND_ITEM**
Appends an item to the end of menu.
Argument: Menu_item
Default: N/A
Procs: set
Objects: Menu
See Also: 11.6.3

**MENU_CLASS**
Gets an enumerated type that identifies the menu class, as set by the package. This may be one of MENU_CHOICE, MENU_COMMAND, or MENU_TOGGLE.
Argument: Menu_class
Default: Defined by package
Procs: get
Objects: Menu

**MENU_CLIENT_DATA**
Specifies an arbitrary value to be attached to a menu or a menu item.
Argument: caddr_t
Default: None
Procs: create, get, set
Objects: Menu, Menu_item
See Also: 11.17.3

**MENU_COLOR**
Specifies the color index to use for the foreground color for a menu or a menu item.
Argument: int
Default: size-1 (where size is the number of colors in the colormap segment associated with the window), for a menu.
Same as the color index specified in the menu item’s menu or for the window, for a menu item.
Procs: create, get, set
Objects: Menu, Menu_item

78 XView Reference Manual
MENU_COL_MAJOR
If TRUE, string items in the menu will be sorted in column-major order (like ls (1)) instead of row-
major order. This attribute does not apply unless the menu uses multiple columns.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Menu

MENU_DEFAULT
Default menu item as a position. The first menu item has position one. Note that a menu title is also a
menu item.

Argument: int
Default: 1, without a menu title; 2, with a menu title
Procs: create, get, set
Objects: Menu

MENU_DEFAULT_ITEM
Default menu item as opaque handle.

Argument: Menu_handle
Default: Handle of first non-title item
Procs: create, get, set
Objects: Menu
See Also: 11.15

MENU_DESCEND_FIRST
If this attribute is specified in xv_find, then the search will be done “depth first.” If it is not speci-
fied, the search will be “deferred”; that is, it will be done horizontally through the menu structure.

Argument: No value
Default: Deferred
Procs: find
Objects: Menu
See Also: 11.14

MENU_DONE_PROC
Specifies a callback procedure that is called when the menu group is dismissed.

Argument: void (*menu_done_proc) ()
Default: menu_return_value(), for a menu object; NULL, for a menu item object
Procs: create, get, set
Objects: Menu
Callback:

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

void

v...
If the menu notify procedure is user-specified, then result is invalid, since user-speci-
fied menu notify procedures do not return a value. If the menu notify procedure is
\texttt{menu\_return\_value()}; then result is the value of the selected menu item. If the
menu notify procedure is \texttt{menu\_return\_item()}, the result is the handle of the se-
lected menu item. You can get notification that any menu in a menu group is done by at-
taching \texttt{MENU\_DONE\_PROC} to each menu. However, you will get better results with
\texttt{menu\_item\_notify} procedures.

\textbf{See Also: 11.4}

\textbf{MENU\_FEEDBACK}

This attribute is for SunView compatibility. For more informations, refer to the manual \textit{Converting
SunView Applications}.

\textbf{MENU\_FIRST\_EVENT}

Gets the event which was initially passed into \texttt{menu\_show}. The event’s contents can be modified.

\textbf{Return Type: Event *}

\textbf{Procs: get}

\textbf{Objects: Menu}

\textbf{See Also: 11.4}

\textbf{MENU\_GEN\_PIN\_WINDOW}

Creates a command window as the pin window based on the menu’s contents. The frame (argument
1) is the parent frame; the name (argument 2) is the pin window’s name. All menu items must have
notify procedures; \texttt{MENU\_NOTIFY\_PROC} for the menu itself is ignored.

\textbf{Argument 1: Frame}

\textbf{Argument 2: char *}

\textbf{Default: No pin window}

\textbf{Procs: create, set}

\textbf{Objects: Menu}

\textbf{See Also: 11.12}

\textbf{MENU\_GEN\_PROC}

Names a client-provided procedure that is called to generate a menu or menu item.

\textbf{Argument: void (**menu\_gen\_proc)()}

\textbf{Default: None}

\textbf{Procs: create, get, set}

\textbf{Objects: Menu, Menu\_item}

\textbf{Callback:}

\begin{verbatim}
Menu
  menu_gen_proc(m, op)
  Menu m;
  Menu_generate op;
\end{verbatim}

This menu generating procedure is called whenever a menu item that has the
\texttt{MENU\_GEN\_PROC} attribute set and the menu needs to be displayed or traversed. It should
return a handle to a menu that has either been dynamically created or statically stored.
The \texttt{op} argument tells the state of the menu when the function is called. The argument \texttt{op}
has one of the following values: \texttt{MENU\_DISPLAY, MENU\_DISPLAY\_DONE, MENU\_NOTIFY,}
or \texttt{MENU\_NOTIFY\_DONE} as defined by \texttt{Menu\_generate} in \texttt{openmenu.h}.

\textbf{See Also: 11.9}

\textbf{MENU\_GEN\_PROC\_IMAGE}

This attribute is for SunView compatibility. For more information, refer to the manual \textit{Converting
SunView Applications}. 

\textbf{XView Reference Manual}
**MENU_GEN_PROC_ITEM**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_GEN_PULLRIGHT**
Defines the generate procedure for the menu item’s submenu.

Argument: `void (*menu_gen_proc)()`
Default: `NULL`
Procs: `create, get, set`
Objects: `Menu_item`
Callback:

```c
Menu menu_gen_proc(m, op)
    Menu m;
    Menu_generate op;
```

This menu generating procedure is called whenever a menu item has `MENU_GEN_PROC` set and the menu needs to be displayed or traversed. It should return a handle to a menu that has either been dynamically created or statically stored.

See Also: 11.9

**MENU_GEN_PULLRIGHT_IMAGE**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_GEN_PULLRIGHT_ITEM**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_IMAGE**
Specifies the menu item’s server image.

Argument: `Server_image`
Default: `NULL`
Procs: `create, get, set`
Objects: `Menu_item`
See Also: 11.6.1

**MENU_IMAGE_ITEM**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_IMAGES**
Creates menu items with the specified server images. The new menu items are appended to the menu.

Argument: `list of Server_image`
Default: `None`
Procs: `create, set`
Objects: `Menu`
See Also: 11.8
MENU_INACTIVE
If TRUE, the menu item is grayed out and not selectable.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Menu_item
See Also: 11.13

MENU_INSERT
Inserts a new menu item after n'th item in the menu.

Argument 1: int
Argument 2: Menu_item
Default: N/A
Procs: create, get, set
Objects: Menu

MENU_INSERT_ITEM
Inserts the menu item given as the next value after the menu item given as the first value.

Argument 1: Menu_item
Argument 2: Menu_item
Procs: create, set
Objects: Menu

MENU_ITEM
Allows you to create menu items in-line with the call to the xv_create() used to create your menu. Takes a menu item-specific attribute-value list that would otherwise be used in a separate call to xv_create() to create menu items with the MENUITEM package.

Argument: A-V list
Procs: create, set
Objects: Menu
Usage:
    xv_create(NULL, MENU,
            MENU_ITEM,
            MENU_STRING, "foo",
            MENU_NOTIFY_PROC, foo_notify_proc,
            NULL,
            MENU_ITEM,
            MENU_STRING, "bar",
            MENU_NOTIFY_PROC, bar_notify_proc,
            NULL,
            NULL);

See Also: 11.6.1

MENU_LAST_EVENT
Gets the last event read by the menu. The event’s contents can be modified.

ReturnType: Event *
Procs: get
Objects: Menu
See Also: 11.4
**MENU_NCOLS**

Specifies the number of columns in a menu.

Argument: int
Default: 1
Procs: create, get, set
Objects: Menu
See Also: 11.11

**MENU_NITEMS**

Returns the number of items in a menu.

Return Type: int
Procs: get
Objects: Menu
See Also: 11.9

**MENU_NOTIFY_PROC**

Names a procedure to be called when the user selects a menu item.

Argument: void (*menu_notify_proc) ()
Default: NULL
Procs: create, get, set
Objects: Menu, Menu_item
Callback:

```c
void menu_notify_proc(Menu menu, Menu_item menu_item) {
    Menu menu;
    Menu_item menu_item;
    // The notify procedure is attached to menus and menu items using MENU_NOTIFY_PROC. This function is called whenever the user selects a menu item. The menu identifies which menu the menu_item belongs to.
}
```

See Also: 11.13

**MENU_NOTIFY_STATUS**

If the menu is attached to a menu button that is part of an unpinned pop-up window, then the window is dismissed if MENU_NOTIFY_STATUS is XV_OK. If MENU_NOTIFY_STATUS is set to XV_ERROR, then the window is not dismissed. You probably will only need to xv_get() or xv_set() this attribute from within a notify procedure. When a notify procedure exits for an unpinned command frame, XView internally uses the value of this attribute to determine whether or not the command frame is dismissed. XView sets the value of this attribute to XV_OK before calling the notify procedure.

Argument: int
Default: XV_OK
Valid Values: [XV_OK, XV_ERROR]
Procs: get, set
Objects: Menu
See Also: 11.16
**MENU_NROWS**
Sets or gets the number of rows in a menu.

Argument: int
Default: The number of menu items in the menu
Procs: create, get, set
Objects: Menu
See Also: 11.11

**MENU_NTH_ITEM**
Gets the nth menu item (n starts at 1).

Return Type: Menu_item
Argument: int
Procs: get
Objects: Menu
See Also: 11.9

**MENU_PARENT**
On a Menu object, this attribute is only valid from within a notify or generate procedure. With a submenu the return value is the Menu Item from which the submenu was pulled-right. For a top-level menu the xv_get() returns NULL. On a Menu_item object, the return value is the handle of the enclosing Menu.

Return Type: Menu or Menu_item
Procs: get
Objects: Menu, Menu_item
See Also: 11.9.1

**MENU_PIN**
Determines whether the menu will have pushpin.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Menu
See Also: 11.12

**MENU_PIN_PROC**
Names a procedure called if a user chooses the pin menu item. The default procedure displays a window whose layout is similar to the menu it replaces.

Argument: void (*menu_pin_proc)()
Default: menu_default_pin_proc()
Procs: create, get, set
Objects: Menu
Callback:

```c
void
menu_pin_proc (menu, x, y)
```
```
Menu menu;
int x, y;
```

This client-supplied procedure is called whenever the user attempts to pin-up a menu. Attached to menus using the attribute, MENU_PIN_PROC. x and y are the fullscreen coordinates of the upper-left corner of the pin window.

See Also: 11.12
**MENU_PIN_WINDOW**
The handle of the command frame representing the pin window for the menu when pinned-up. Use **MENU_GEN_PIN_WINDOW** to have XView manage this frame automatically. When using **MENU_GEN_PIN_WINDOW**, the value returned by **xv_get()** for **MENU_PIN_WINDOW** will be NULL until a menu is initially pinned.

- **Argument:** Frame_cmd
- **Default:** None
- **Procs:** create, get, set
- **Objects:** Menu
- **See Also:** 11.12

**MENU_PULLRIGHT**
Item’s pullright menu.

- **Argument:** Menu
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Menu_item
- **See Also:** 11.8

**MENU_PULLRIGHT_IMAGE**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_PULLRIGHT_ITEM**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**MENU_RELEASE**
Specifies that the menu item gets destroyed when its parent menu is destroyed. This is the default for menu items created in-line.

- **Argument:** No value
- **Default:** Destroy item, if in-line; do not destroy item if Append or Replace.
- **Procs:** create, set
- **Objects:** Menu_item
- **See Also:** 11.17

**MENU_RELEASE_IMAGE**
Specifies that the string or Server_image associated with the item is freed when the item is destroyed.

- **Argument:** No value
- **Default:** Do not release text string or server image.
- **Procs:** create, set
- **Objects:** Menu_item
- **See Also:** 11.8

**MENU_REMOVE**
Removes the nth item from the menu (n starts at 1).

- **Argument:** int
- **Procs:** set
- **Objects:** Menu
- **See Also:** 11.9
MENU_REMOVE_ITEM
Removes the specified menu item.
Argument: Menu_item
Procs: set
Objects: Menu

MENU_REPLACE
Replicates the n-th menu item (argument 1) with the menu item specified in argument 2 (n starts at 1).
Argument 1: int
Argument 2: Menu_item
Procs: create, set
Objects: Menu

MENU_REPLACE_ITEM
Replicates the menu item given as first value with the one given as the second value in the menu (the old item is not replaced in any other menus in which it may appear).
Argument 1: Menu_item
Argument 2: Menu_item
Procs: create, set
Objects: Menu

MENU_SELECTED
This attribute is valid for both a menu and a menu item. On a menu, it returns the selected menu item number. On a menu item, it returns TRUE if the item is selected, otherwise it returns FALSE.
Argument: int or Bool
Default: FALSE
Procs: get
Objects: Menu, Menu_item
See Also: 11.15

MENU_SELECTED_ITEM
Returns the selected menu item.
Argument: Menu_item
Default: None
Procs: get
Objects: Menu
See Also: 11.15

MENU_STRING
Sets or gets the menu item’s string. The string is not copied when this attribute is set.
Argument: char *
Default: NULL
Procs: create, get, set
Objects: Menu_item
See Also: 11.6.1

MENU_STRING_ITEM
This attribute is for SunView compatibility. For more information, refer to the manual Converting SunView Applications.
MENU_STRINGS

Creates menu items with the specified strings. The new menu items are appended to the menu. The
strings are not copied when this attribute is set.

Argument:  list of char *
Procs:      create, set
Objects:    Menu
See Also:   11.5

MENU_STRINGS_AND_ACCELERATORS

Creates menu items with a given label and accelerator. If an accelerator is changed with set,
FRAME_MENUS must be set again before the accelerator will take effect (see FRAME_MENUS).

The accelerator strings will be copied by XView. The menu item label strings will not be copied by
XView.

Argument:  Null terminated list of label(char *), accelerator(char *) pairs
Default:   None
Procs:     create, set
Objects:   Menu

```
xv_set(menu, MENU_NOTIFY_PROC, file_proc,
       MENU_STRINGS_AND_ACCELERATORS,
       "Load", "Meta+l",
       "Print", "Meta+p",
       "Include","Meta+Ctrl+i",
       NULL);
```

MENU_TITLE

Specifies that the item is the menu’s title. Returns TRUE or FALSE on get.

Return Type:  Bool
Argument:     No value
Default:      FALSE
Procs:        create, get, set
Objects:      Menu_item

MENU_TITLE_ITEM

Creates a string title item. For OPEN LOOK compliance, this attribute must not be used with menus
that originate from pullright items or pulldown menu buttons. The string is not copied when this attribute
is set.

Argument:    char *
Procs:        create, set
Objects:      Menu
See Also:     11.5

MENU_TYPE

Returns MENU_MENU or MENU_ITEM. Informs you whether the object is a menu or a menu item.

Argument:    Menu_attribute (an enum)
Default:     MENU_MENU
Procs:       get
Objects:     Menu, Menu_item
MENU_VALID_RESULT
If TRUE, then a zero return value represents a legitimate value. This attribute is used only when the menu notify procedure that is invoked is the public procedure menu_return_value().

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Menu

MENU_VALUE
Sets or gets the item’s value. This attribute is used only when the menu notify procedure that is invoked is the public procedure menu_return_value().

Argument: Xv_opaque
Default: NULL
Procs: create, get, set
Objects: Menu_item

NOTICE_BLOCK_THREAD
Specifies the type of notice desired. If TRUE, the notice, when mapped (via XV_SHOW), will block the thread of execution.

Argument: Bool
Default: TRUE
Procs: create, set
Objects: Xv_Notice
See Also: 12.2

NOTICE_BUSY_FRAMES
Specifies the frames or windows to appear busy during notice pop-up. This applies only when NOTICE_LOCK_SCREEN is FALSE.

Argument: NULL-terminated list, Frame
Default: NULL
Procs: create, set
Objects: Xv_Notice
See Also: 12.2.1

NOTICE_BUTTON
Specifies a string to be displayed in a button and a value to use if the button is selected.

Return Type: int (notice_prompt() returns the value of the button selected)
Argument 1: char *
Argument 2: int
Default: None
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.1.1
**NOTICE_BUTTON_NO**

Specifies a string associated with the NO button. The value returned if this button is selected is NOTICE_NO.

Return Type: int (notice_prompt() returns the value of the button selected)
Argument: char *
Default: None
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.1.1

**NOTICE_BUTTON_YES**

Specifies a string to associate with the YES (confirm) button. The value returned when this button is selected is NOTICE_YES.

Return Type: int (notice_prompt() returns the value of the button selected)
Argument: char *
Default: None
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.1.1

**NOTICE_EVENT_PROC**

Specifies the function to be called when a notice button is clicked on. This applies only when NOTICE_LOCK_SCREEN is FALSE.

Argument: void (*my_notice_proc)()
Default: NULL
Procs: create, set
Objects: Xv_Notice
Callback:

```c
void
my_notice_proc(notice, value, event)
    Xv_Notice *notice;
    int value;
    Event *event;
```

See Also: 12.2.1.1

**NOTICE_FOCUS_XY**

Specifies the x, y position from which the notice shadow emanates. The x, y position is relative to the owner window. This applies to a notice object only when NOTICE_LOCK_SCREEN is TRUE.

Argument 1: int
Argument 2: int
Default: Current mouse position
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.2.2
NOTICE_FONT
Specifications the font to be used in the notice.
Argument: Xv_Font
Default: The font of the owner frame
Procs: create, get, notice_prompt()
Objects: Xv_Notice

NOTICE_LOCK_SCREEN
Specifications the type of notice desired. If TRUE, the notice locks up the screen when mapped.
Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Xv_Notice
See Also: 12.2

NOTICE_MESSAGE_STRING
Specifications the text to print in a notice. The argument is a single NULL-terminated string, which may contain \n as a line break.
Argument: (char *)
Default: NULL
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.1

NOTICE_MESSAGE_STRINGS
Specifications the text to print in a notice. Argument is a NULL-terminated list of strings, which may contain \n as a line break.
Argument: list of char *
Default: NULL
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.1

NOTICE_MESSAGE_STRINGS_ARRAY_PTR
Specifications the text to print in a notice. The argument is a variable pointing to a NULL-terminated array of strings, which may contain \n as a line break.
Argument: char **
Default: NULL
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 24.4.2, 12.1

NOTICE_NO_BEEPING
Allows a client to specify that no beeping should take place, regardless of the default resource database setting. The default for this option is FALSE; that is, beep the number of times specified in the database.
Argument: int
Default: TRUE
Procs: create, get, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.2.2.1
NOTICE_STATUS
Specifies the address of the return code of the notice when it pops-down. If not specified, the notice return code is stored in the notice object and can be obtained by doing \((\text{int})xv\_get(\text{NOTICE}\_\text{STATUS})\). In \text{xv_set()} we pass in an \((\text{int}\ *)\), and the \text{xv_get()} returns an \((\text{int})\).

Return Type: \((\text{int})\)
Argument: \((\text{int}\ *)\)
Default: See description
Procs: create, get, set
Objects: Xv_Notice
See Also: 12.1.1

NOTICE_TRIGGER
Specifies an XView event ID or type, other than clicking on mouse buttons, that will cause the notice to pop down. The notice pops down and the thread of execution continues. This applies to a notice object only when NOTICE_LOCK_SCREEN is TRUE. When this event occurs, the value returned is NOTICE_TRIGGERED. The event parameter to notice_prompt contains specifics about the event that triggered it.

Argument: \(\text{int}\) (for example, \text{MS\_LEFT})
Default: N/A
Procs: create, set, notice_prompt()
Objects: Xv_Notice
See Also: 12.2.2.1, NOTICE_TRIGGER_EVENT

NOTICE_TRIGGER_EVENT
Specifies the address of the XView Event structure that caused the notice to pop down, the thread of execution continues. This is used with the NOTICE_TRIGGER attribute and applies only when NOTICE_LOCK_SCREEN is TRUE.

Argument: \((\text{Event}\ *)\)
Default: N/A
Procs: create, set
Objects: Xv_Notice
See Also: 12.2.2.1, NOTICE_TRIGGER

OPENWIN_ADJUST_FOR_HORIZONTAL_SCROLLBAR
Reserves space in openwin-class objects (e.g., canvas) for a horizontal scrollbar. On \text{xv_set}, adjusts (extends) the height of a subwindow that does not have a horizontal scrollbar to align properly with one that does.

Argument: \text{Bool}
Default: \text{TRUE}
Procs: create, get, set
Objects: Openwin

OPENWIN_ADJUST_FOR_VERTICAL_SCROLLBAR
Reserves space in openwin-class objects (e.g., canvas) for a vertical scrollbar. On \text{xv_set}, adjusts (extends) the width of a subwindow that does not have a vertical scrollbar to align properly with one that does.

Argument: \text{Bool}
Default: \text{TRUE}
Procs: create, get, set
Objects: Openwin
OPENWIN_AUTO_CLEAR

If TRUE, exposed areas of windows are cleared (i.e., painted the background color) before the repaint procedure is called.

Argument: Bool
Default: TRUE unless subwindow’s canvas is retained
Procs: create.get.set
Objects: Openwin
See Also: 5.3

OPENWIN_HORIZONTAL_SCROLLBAR

Returns the handle of the horizontal scrollbar associated with the specified view.

Return Type: Scrollbar
Argument: Xv_Window
Procs: create.get.set
Objects: Openwin
Usage:

Scrollbar sb;
Xv_Window view;
Openwin openwin;

sb = (Scrollbar)xv_get(openwin,
OPENWIN_HORIZONTAL_SCROLLBAR, view);

See Also: OPENWIN_VERTICAL_SCROLLBAR

OPENWIN_NO_MARGIN

If TRUE, the view window’s two pixel bottom and right margins are turned off.

Argument: Bool
Default: FALSE
Procs: create.get.set
_objects: Openwin

OPENWIN_NTH_VIEW

Gets the handle of a specified openwin view window. Openwin view windows are numbered from zero.

Return Type: Xv_window
Argument: int
Procs: get
_objects: Openwin
See Also: 5.6.3.2

OPENWIN_NVIEWS

Gets the number of views contained in the open window.

Return Type: int
Default: 1
Procs: get
_objects: Openwin
OPENWIN_SHOW_BORDERS
Displays openwin borders. This must remain on for openwin objects to have scrollbars attached to them.

Argument: Bool
Default: TRUE (FALSE for scrollable panels)
Procs: create, get
Objects: Openwin

OPENWIN_SPLIT
Takes as its value a list of attribute-value pairs beginning with OPENWIN_SPLIT_.

Argument: A-V list
Procs: create, set
Objects: Openwin

Callback:

```c
void
openwin_split_destroy_proc(openwin)
Openwin openwin;

OPENWIN_SPLIT_ used to install this procedure. This function is called when the user
joins two views.
```

See Also: 5.6.2

OPENWIN_SPLIT_DESTROY_PROC
Names a procedure to call when a split openwin is joined. This attribute can only be set in an
OPENWIN_SPLIT attribute-value list.

Argument: void (*openwin_split_destroy_proc) ()
Default: NULL
Procs: create, get, set
Objects: Openwin

See Also: 5.6.3.1, OPENWIN_SPLIT

OPENWIN_SPLIT_DIRECTION
Sets the direction of the split either horizontally or vertically. This attribute can only be set in an
OPENWIN_SPLIT attribute-value list.

Argument: Openwin_split_direction
Default: OPENWIN_SPLIT_HORIZONTAL
Procs: create, set
Objects: Openwin

See Also: OPENWIN_SPLIT

OPENWIN_SPLIT_INIT_PROC
Names a procedure to call when a split window is created. This attribute can only be set in an
OPENWIN_SPLIT attribute-value list.

Argument: void (*openwin_split_init_proc) ()
Default: NULL
Procs: create, get, set
Objects: Openwin

See Also: 5.6.3.1, OPENWIN_SPLIT
OPENWIN_SPLIT_POSITION
Sets the position (in pixels) of the view. This attribute can only be set in an OPENWIN_SPLIT attribute-value list.

Argument: int
Default: None
Procs: create, set
Objects: Openwin
See Also: OPENWIN_SPLIT

OPENWIN_SPLIT_VIEW
Specifies which view to split. Its value is the handle of the view you want to split. This attribute can only be set in an OPENWIN_SPLIT attribute-value list.

Argument: Xv_window
Procs: set
Objects: Openwin
See Also: OPENWIN_SPLIT

OPENWIN_SPLIT_VIEW_START
Specifies which part of the data (measured in scrollbar-units) is displayed at the start of the view (top for vertical; left for horizontal). This attribute can only be set in an OPENWIN_SPLIT attribute-value list.

Argument: int
Default: Continue from previous view
Procs: create, get, set
Objects: Openwin
See Also: OPENWIN_SPLIT

OPENWIN_VERTICAL_SCROLLBAR
Returns the handle of the vertical scrollbar associated with the specified view.

Return Type: Scrollbar

Argument: Xv_Window
Procs: create, get, set
Objects: Openwin
Usage:

Scrollbar sb;
Xv_Window view;
Openwin openwin;

sb = (Scrollbar)xv_get(openwin,
OPENWIN_VERTICAL_SCROLLBAR, view);

See Also: OPENWIN_HORIZONTAL_SCROLLBAR

OPENWIN_VIEW_ATTRS
Distributes modifications across all views in a given openwin. Note that this does not affect canvas paint windows.

Argument: A-V list
Procs: create, set
Objects: Openwin
See Also: CANVAS_PAINTWINDOW_ATTRS
**PANEL_ACCEPT_KEYSTROKE**

Specifies whether the panel background or panel item should consume keyboard events. When an item wants keystrokes, set `PANEL_ACCEPT_KEYSTROKE` to `TRUE` on the panel item in the item’s init routine. Setting `PANEL_ACCEPT_KEYSTROKE` to `TRUE` on the panel background is not recommended, since this is not supported by OPEN LOOK. This mode is maintained for SunView1 compatibility.

**Warning:** this attribute should *only* be used from within a panel item extension.

Argument: `Bool`

Default: `FALSE` for panel background. Default for panel items depends on the state of `OpenWindows.KeyboardCommands`. If it is SunView1 or Basic, then only `PANEL_TEXT`, `PANEL_NUMERIC_TEXT`, and `PANEL_MULTILINE_TEXT` accept keyboard input focus. If it is Full, then all panel items accept keyboard input focus.

Procs: `create`, `get`, `set`

Objects: `Panel`, `Panel_item`

See Also: 7.19.6

---

**PANEL_BACKGROUND_PROC**

Names an event-handling procedure called when an event falls on the background of the panel.

Argument: `void (*panel_background_proc) ()`

Default: `panel_default_handle_event`

Procs: `create`, `get`, `set`

Objects: `Panel`

See Also: 7.19.6

---

**PANEL_BORDER**

Adds a border around the panel. In a 3D implementation, this border is two pixels wide and presents a “chiseled” appearance. In 2D, the border is one pixel wide. Since the border is rendered directly on the Panel, it is the job of the application to make sure that no `Panel_item`’s overlap the border.

Argument: `Bool`

Default: `FALSE`

Procs: `create`, `get`, `set`

Objects: `Panel`

See Also: 7.3.2

---

**PANEL_BLINK_CARET**

This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*. 

---

*XView Attributes* 95
**PANEL_BUSY**

Sets a button’s or a drop target item’s busy state. While a button or drop target is busy, it will not accept further input (e.g., SELECT-down). By default, a button or Drop Target Item will be in the busy state while in its notify procedure and be cleared upon exiting. The busy state can be maintained after exiting the notify procedure by setting PANEL_BUSY to TRUE from within the notify procedure. Setting PANEL_BUSY back to FALSE, at a later time, clears the button’s busy state. This attribute can also be used for similar functionality in panel item extensions.

- **Argument:** Bool
- **Default:** Set to FALSE prior to entering button’s or drop target’s notify procedure
- **Procs:** get, set
- **Objects:** Panel_button_item, Panel_item, Panel_drop_target_item

See Also: 7.9.1

**PANEL_CARET_ITEM**

Specifies the panel item that currently has the input focus.

- **Argument:** Panel_item
- **Default:** First item that can accept keyboard input
- **Procs:** create, get, set
- **Objects:** Panel

See Also: 7.15.1

**PANEL_CHILD_CARET_ITEM**

Specifies what embedded (child) panel item to set the keyboard focus to when the application sets PANEL_CARET_ITEM to the parent panel item. PANEL_CHILD_CARET_ITEM should be NULL, if there are no embedded (child) panel items, or if the parent panel item itself can take the keyboard focus.

- **Argument:** Panel_item
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Panel_item with an embedded (child) panel item.

**PANEL_CHOICE_COLOR**

Specifies the foreground color index (argument 2) for the specified choice (argument 1).

- **Argument 1:** int (choice index)
- **Argument 2:** int (color_index)
- **Default:** Panel foreground color
- **Procs:** create, get, set
- **Objects:** Panel_choice_item
- **Usage:**

  ```
  xv_set(pchoice, PANEL_CHOICE_COLOR, 1, RED, NULL);
  ```

See Also: 7.10.6, 21.5

**PANEL_CHOICE_FONT**

This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**PANEL_CHOICE_FONTS**

This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.
**PANEL_CHOICE_IMAGE**

Specifies the image (argument 2) for the specified choice (argument 1).

- **Argument 1:** int
- **Argument 2:** server_image
- **Default:** None
- **Procs:** create, get, set
- **Objects:** Panel_choice_item
- **See Also:** 7.10.2

**PANEL_CHOICE_IMAGES**

Specifies the image for each of several choices.

- **Argument:** NULL-terminated list of Server_images
- **Default:** NULL
- **Procs:** create, set
- **Objects:** Panel_choice_item
- **See Also:** 7.10

**PANEL_CHOICE_NCOLS**

Specifies the number of columns to use in the layout of panel choices.

- **Argument:** int
- **Default:** 1 for vertical layout (PANEL_LAYOUT)
- **Procs:** create, get, set
- **Objects:** Panel_choice_item
- **See Also:** 7.10.1

**PANEL_CHOICE_NROWS**

Specifies the number of rows to use in the layout of panel choices.

- **Argument:** int
- **Default:** 1 for horizontal layout (PANEL_LAYOUT)
- **Procs:** create, get, set
- **Objects:** Panel_choice_item
- **See Also:** 7.10.1

**PANEL_CHOICE_RECT**

The rectangle that encloses the specified choice. The first argument is the index of the choice (0 = first choice). This attribute is not valid for PANEL_CHOICE_STACK objects.

- **Argument:** int
- **Return Type:** Rect *
- **Procs:** get
- **Objects:** Panel_choice_item
- **See Also:** 7.10.7

**PANEL_CHOICE_STRING**

Specifies the string (argument 2) for the specified choice (argument 1). The string is copied when this attribute is set.

- **Argument 1:** int
- **Argument 2:** char *
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Panel_choice_item
- **See Also:** 7.10
**PANEL_CHOICE_STRINGS**
Specifies the string for each choice. You must specify at least one choice. The least you can specify is a single NULL-terminated string. The strings are copied when this attribute is set.

Argument: NULL-terminated list of char *
Default: NULL
Procs: create, set
Objects: Panel_choice_item
See Also: 7.10, 25.2.1

**PANEL_CHOICE_X**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**PANEL_CHOICE_XS**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**PANEL_CHOICE_Y**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**PANEL_CHOICE_YS**
This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**PANEL_CHOOSE_NONE**
Allows scrolling lists or choice items to have no currently selected item. Not applicable if PANEL_CHOOSE_ONE is FALSE.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Panel_choice_item, Panel_list_item
See Also: 7.10.2

**PANEL_CHOOSE_ONE**
If TRUE, creates an exclusive scrolling list or choice. If FALSE, creates a non-exclusive scrolling list or choice. PANEL_CHOOSE_ONE is used as part of the PANEL_TOGGLE macro. When creating a toggle, it is recommended that you use the macro instead of the attribute.

Argument: Bool
Default: TRUE
Procs: create, get
Objects: Panel_choice_item, Panel_list_item
See Also: 7.11.3

**PANEL_CLIENT_DATA**
Specifies an arbitrary value to be attached to a panel or to individual panel items.

Argument: caddr_t
Default: None
Procs: create, get, set
Objects: Panel, Panel_item
See Also: 7.19.1
**PANEL_CURRENT_ITEM**

Returns the handle of the currently active panel item. An item is considered current when there is a mouse button down event pending on that item (for example, after a SELECT-down over a text scrolling button, but before the corresponding SELECT-up). NULL implies no panel item is currently active.

Argument: Panel_item
Procs: get
Objects: Panel

**PANEL_DEFAULT_ITEM**

Sets the default panel item in the panel or gets the handle of the default panel item. Only buttons are valid as default items. Application programmers use this attribute to indicate the default button in a popup frame. Then, using the DefaultAction key, usually RETURN, causes the default button to be activated.

Argument: Panel_item
Default: NULL
Procs: create, get, set
Objects: Panel

**PANEL_DEFAULT_VALUE**

Identifies the default choice in PANEL_CHOICE_STACK items.

Argument: int or unsigned
Default: First choice
Procs: create, get, set
Objects: Panel_choice_item

**PANEL_DIRECTION**

Identifies the horizontal or vertical orientation of a slider or gauge.

Argument: Panel_setting
Valid Values: \{PANEL_HORIZONTAL, PANEL_VERTICAL\}
Default: PANEL_HORIZONTAL
Procs: create, get, set
Objects: Panel_slider_item, Panel_gauge_item
See Also: 7.13

**PANEL_DISPLAY_LEVEL**

Specifies the number of choices to display. Values are one of PANEL_ALL, PANEL_CURRENT, or PANEL_NONE. PANEL_DISPLAY_LEVEL is used as part of the PANEL_CHOICE_STACK macro; it is recommended that you use the macro instead of the attribute.

Argument: Panel_setting
Valid Values: \{PANEL_ALL, PANEL_CURRENT, PANEL_NONE\}
Default: PANEL_ALL
Procs: create, get
Objects: Panel_choice_item
See Also: 7.10.1
PANEL_DISPLAY_ROWS
Number of rows to display in the multiline text field.

Argument: int
Default: 5
Procs: create,get,set
Objects: Panel_multiline_text_item
See Also: 7.17

PANEL_DROP_BUSY_GLYPH
Busy drop target glyph, as defined by drag and drop specification.

Argument: Server_image
Default: Use normal drop target glyph
Procs: create,get,set
Objects: Panel_drop_target_item
See Also: 7.18.1.2

PANEL_DROP_DND
Drag and drop object (DRAGDROP) associated with panel drop target item. The DRAGDROP object is used to initiate a drag and drop operation. If no PANEL_DROP_DND exists, then the panel Drop Target Item does not support drags, and is termed an "empty" drop target (PANEL_DROP_FULL = FALSE).

Argument: Drag_drop
Default: NULL
Procs: create,get,set
Objects: Panel_drop_target_item
See Also: 7.18.1.3

PANEL_DROP_FULL
If TRUE, then the drop target item has draggable data set on the PANEL_DROP_DND object’s selection items. The normal glyph is displayed. Setting PANEL_DROP_FULL to TRUE when no PANEL_DROP_DND is defined is not valid; PANEL_DROP_FULL will remain FALSE. If FALSE, then no draggable data is available (i.e., the drop target is empty). No glyph is displayed.

Argument: Bool
Default: FALSE
Procs: create,get,set
Objects: Panel_drop_target_item
See Also: 7.18.1.3

PANEL_DROP_GLYPH
Normal drop target glyph, as defined by drag and drop package.

Argument: Server_image
Default: None
Procs: create,get,set
Objects: Panel_drop_target_item
See Also: 7.18.1.2

PANEL_DROP_HEIGHT
Dimensions of drag target box, in pixels, excluding border and margin.

Argument: int
Default: 16
Procs: create,get,set
Objects: Panel_drop_target_item
**PANEL_DROP_SEL_REQ**
Selection requestor object (SELECTION_REQUESTOR) associated with the panel drop target item. The Selection_requestor object is used to receive a drop.

Argument: Selection_requestor
Procs: get
Objects: Panel_drop_target_item
See Also: 7.18.1.4

**PANEL_DROP_SITE_DEFAULT**
Specifies whether or not the panel drop target’s drop site is the default drop site. Only one panel drop target in a frame may be the default drop target. See DROP_SITE_DEFAULT attribute for more information.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Panel_drop_target_item

**PANEL_DROP_WIDTH**
Dimensions of drag target box, in pixels, excluding border and margin.

Argument: int
Default: 12
Procs: create, get, set
Objects: Panel_drop_target_item

**PANEL_EVENT_PROC**
Event handler for panel items.

Argument: void (*panel_event_proc) ()
Default: Panel_item specific.
Procs: create, get, set
Objects: Panel_item
Callback:

```c
void panel_event_proc(item, event)
    Panel_item item
    Event *event
```
Client-specified event-handling procedure for handling events on panel items. This procedure is installed using PANEL_EVENT_PROC.

See Also: 7.19.7

**PANEL_EXTRA_PAINT_HEIGHT**
Defines the increment by which the panel grows in the y direction. It is used when window_fit_height() is called.

Argument: int
Default: 1 pixel
Procs: create, get, set
Objects: Panel
See Also: 7.5
**PANEL_EXTRA_PAINT_WIDTH**
Defines the increment by which the panel grows in the x direction. It is used when window_fit_width() is called.

Argument: int  
Default: 1 pixel  
Procs: create, get, set  
Objects: Panel  
See Also: 7.5

**PANEL_FEEDBACK**
Specifies feedback to give when an item is selected. If PANEL_DISPLAY_LEVEL is PANEL_CURRENT, the default value is PANEL_NONE; otherwise, it is PANEL_MARKED. PANEL_FEED_BACK is used as part of the PANEL_CHECK_BOX macro. It is recommended that you use the macro instead of the attribute.

Argument: Panel_setting  
Default: See description  
Procs: create, get, set  
Objects: Panel_choice_item

**PANEL_FIRST_ITEM**
Gets the handle of the first item on a panel. The PANEL_EACH_ITEM macro (see Section 3, Procedures and Macros) can be used to iterate over each item in a panel, starting with this first item.

Argument: Panel_item  
Procs: get  
Objects: Panel  
See Also: 7.7

**PANEL_FIRST_PAINT_WINDOW**
Returns a pointer to the first Panel_paint_window struct, which defines the paint window and view window associated with the first view in the scrollable panel.

Argument: Panel_paint_window  
Procs: get  
Objects: Scrollable_panel

**PANEL_FOCUS_PW**
The paint window which currently has or last had the input focus. The has_input_focus Panel_status flag indicates whether or not the scrollable panel currently has the input focus. Initially, the focus paint window is set to the first paint window created in the scrollable panel.

Warning: This attribute should only be used from within a panel item extension.

Argument: Xv_Window  
Procs: get  
Objects: Scrollable_panel

**PANEL_GAUGE_WIDTH**
Specifies the length of the panel gauge “thermometer” bar, in pixels, regardless of its horizontal or vertical orientation.

Argument: int  
Default: 100  
Procs: create, get, set  
Objects: Panel_gauge_item  
See Also: 7.14
**PANEL_GINFO**

The OLGX Graphics Information structure for the specified panel or panel item. This attribute is only of interest to those panel item extensions that are calling OLGX functions or using OLGX macros.

**Warning:** This attribute should *only* be used from within a panel item extension.

**Argument:** Graphics_info *

**Procs:** get

**Objects:** Panel, Panel_item

---

**PANEL_INACTIVE**

If TRUE, panel item cannot be selected. Inactive items are displayed with gray-out pattern.

**Argument:** Bool

**Default:** FALSE

**Procs:** create, get, set

**Objects:** Panel_item

**See Also:** 7.9.1.1

---

**PANEL_ITEM_CLASS**

Gets the panel item *type* of the panel item specified.

**Argument:** Panel_item_type

**Procs:** get

**Objects:** Panel_item

**See Also:** 7.9.1.1

---

**PANEL_ITEM_COLOR**

Specifies the colormap index to use for the item. A value of -1 implies the foreground color for the panel window.

**Argument:** int

**Default:** -1

**Procs:** create, get, set

**Objects:** Panel_item

**See Also:** 21.5.1

---

**PANEL_ITEM_CREATED**

The Panel_item has finished processing the XV_END_CREATE phase of the generic panel item package (*i.e.* xv_panel_item_pkg). All Panel_items are subclassed from xv_panel_item_pkg.

**Warning:** This attribute should *only* be used from within a panel item extension.

**Argument:** Bool

**Procs:** get

**Objects:** Panel_item

---

**PANEL_ITEM_DEAF**

The item does not want any events. This is used for items that contain one or more windows that are located within the item’s rectangle. These windows are interposed on in order to process their events and maintain the panel’s internal variables. An example of this is PANEL_MULTILINE_TEXT.

**Warning:** This attribute should *only* be used from within a panel item extension.

**Argument:** Bool

**Procs:** get, set

**Objects:** Panel_item
PANEL_ITEM_LABEL_RECT

The rectangle describing the label portion of an item.

Warning: This attribute should only be used from within a panel item extension.

Argument: Rect *
Proc: get, set
Objects: Panel_item

PANEL_ITEM_MENU

Specifies the menu associated with the panel item.

Argument: Menu
Default: NULL
Proc: create, get, set
Objects: Panel_item

PANEL_ITEM_NTH_WINDOW

Returns the nth embedded window that receives events in the specified panel item. The first embedded window has index 0. This attribute functions exactly like OPENWIN_NTH_VIEW. This attribute is an “advanced” attribute; the general application programmer will not need to use this attribute.

Return Type: Xv_Window
Argument: int
Proc: get
Objects: Panel_item
Usage:

    textsw_view = xv_get(mltxt_item,
                        PANEL_ITEM_NTH_WINDOW, 0, NULL);

See Also: OPENWIN_NTH_VIEW

PANEL_ITEM_NWINDOWS

Returns the number of embedded windows in the specified panel item that receives events. Zero indicates that there are no embedded windows. This attribute functions exactly like OPENWIN_NVIEWS. This attribute is an “advanced” attribute; the general application programmer will not need to use this attribute.

Return Type: int
Proc: get
Objects: Panel_item
See Also: OPENWIN_NVIEWS

PANEL_ITEM_OWNER

Allows the client to obtain the parent object given the child. In essence, if PANEL_ITEM_OWNER ever returns a value (non-NULL), then the client should assume this is the proper handle.

Values retrieved or set on the child handle are not guaranteed to be forward compatible if those values are not handled through the parent object’s API.

Argument: XV_object
Proc: get
Objects: Panel_item
PANEL_ITEM_RECT
Gets the rectangle surrounding the panel item.

Argument: Rect *
Procs: get
Objects: Panel_item

PANEL_ITEM_VALUE_RECT
The rectangle describing the value portion of an item.
Warning: This attribute should only be used from within a panel item extension.

Argument: Rect *
Procs: get, set
Objects: Panel_item

PANEL_ITEM_WANTS_ADJUST
The panel item wants ACTION_ADJUST events.
Warning: This attribute should only be used from within a panel item extension.

Argument: Bool
Procs: get, set
Objects: Panel_item

PANEL_ITEM_WANTS_ISO
This flag indicates that the panel item wants all ISO characters. Unmodified ASCII characters are not to be interpreted as mouseless keyboard commands. Currently, PANEL_MULTILINE_TEXT and PANEL_TEXT items have this flag set.
Warning: This attribute should only be used from within a panel item extension.

Argument: Bool
Default: FALSE
Procs: get, set
Objects: Panel_item

PANEL_ITEM_X
Specifies the x position (in pixels) where the last panel item was created. If no item was created, then it specifies the x position where the first item will be created.

Argument: int
Procs: get
Objects: Panel
See Also: 7.4.2

PANEL_ITEM_X_GAP
When set on a panel, the horizontal space, in pixels, between the last panel item created and the next panel item; when set on a panel item it is the horizontal space, in pixels, between the last panel item created and this panel item. This attribute allows applications to more easily use relative item positioning instead of absolute positioning. Relative item positioning is the recommended method of positioning panel items, since the size of items may change between invocations (e.g., different scales) or versions of the toolkit.

Argument: int
Default: 10, for Panel; the panel’s PANEL_ITEM_X_GAP, for panel item
Procs: create, get, set, for panel; create, get, for panel item
Objects: Panel, Panel_item
See Also: 7.3.1
**PANEL_ITEM_X_POSITION**

The default x coordinate of the panel item.

**Warning:** This attribute should *only* be used from within a panel item extension.

Argument: int  
Procs: get, set  
Objects: Panel

**PANEL_ITEM_Y**

Specifies the y position (in pixels) where the last panel item was created. If no item was created, then it specifies the y position where the first item will be created.

Argument: int  
Default: None  
Procs: get  
Objects: Panel  
See Also: 7.4.2

**PANEL_ITEM_Y_GAP**

When set on a panel, this is the vertical space, in pixels, between the last panel item created and the next panel item when set on a panel item, PANEL_ITEM_Y_GAP is the vertical space, in pixels, between the last panel item created and this panel item. This attribute allows applications to more easily use relative item positioning instead of absolute positioning. Relative item positioning is the recommended method of positioning panel items, since the size of items may change between invocations (e.g., different scales) or versions of the toolkit.

Argument: int  
Default: 13 for panel, the panel's PANEL_ITEM_Y_GAP for panel item  
Procs: create, get, set for panel; create, get for panel item  
Objects: Panel, Panel_item  
See Also: 7.3.1

**PANEL_ITEM_Y_POSITION**

The default y coordinate of the panel item.

**Warning:** This attribute should *only* be used from within a panel item extension.

Argument: int  
Procs: get, set  
Objects: Panel

**PANEL_JUMP_DELTA**

Specifies the number of client units to adjust the value when a JumpLeft or JumpRight keyboard command is issued for a horizontal slider. Also specifies the number of client units to adjust the value when a JumpUp or JumpDown keyboard command is issued for a vertical slider or numeric text item. Used in conjunction with the OPEN LOOK Mouseless Model.

Argument: int  
Default: 10  
Procs: create, get, set  
Objects: Panel_slider_item, Panel_numeric_text_item
PANEL_LABEL_BOLD

If TRUE, PANEL_MESSAGE_ITEM's label is rendered in bold.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Panel_message_item
See Also: 7.12

PANEL_LABEL_FONT

Defines the font to use in the label portion of a panel item.

Warning: Use of this attribute may cause your application to violate the OPEN LOOK Graphical User Interface Functional Specification.

Argument: Xv_Font
Default: Panel font
Procs: create, get, set
Objects: Panel_item

PANEL_LABEL_IMAGE

Specifies the image for an item’s label.

Argument: Server_image
Default: NULL
Procs: create, get, set
Objects: Panel_item
See Also: 15.4.1

PANEL_LABEL_STRING

Specifies the string for the item’s label. The string is copied when this attribute is set. By using \n, newlines can be placed into the strings to generate multi-line labels.

Argument: char *
Default:"
Procs: create, get, set
Objects: Panel_item
Usage:

xv_create(panel, PANEL_TEXT,
  PANEL_LABEL_STRING, "Multi\nLine\nLabel:",
  NULL);

See Also: 7.9, 7.3.2

PANEL_LABEL_WIDTH

Specifies the width of the panel item label in pixels. For buttons, PANEL_LABEL_WIDTH does not include endcaps and menu marks (if any are present). PANEL_LABEL_WIDTH has no effect on a panel button until the panel button’s PANEL_LABEL_STRING or PANEL_LABEL_IMAGE is set.

Argument: int
Default: Width of text string or server image
Procs: create, get, set
Objects: Panel_item
See Also: 7.9.3
**PANEL_LABEL_X**

Specifies the x coordinate of the label portion of panel items that have values associated with them (e.g., text items). Intended to be used when PANEL_VALUE_X is specified.

**Argument:** int (in pixels)

**Default:** PANEL_ITEM_X from parent panel

**Procs:** create, get, set

**Objects:** Panel_item

**See Also:** PANEL_VALUE_X

**PANEL_LABEL_Y**

Specifies the y coordinate of the label portion of panel items that have values associated with them (e.g., text items). Intended to be used only if PANEL_VALUE_Y is specified.

**Argument:** int (in pixels)

**Default:** PANEL_ITEM_Y from parent panel

**Procs:** create, get, set

**Objects:** Panel_item

**See Also:** PANEL_VALUE_Y

**PANEL_LAYOUT**

In a PANEL create call, this attribute controls the layout of panel items. When used in a create panel item call, this attribute controls the direction in which the item’s components are laid out.

**Argument:** Panel_setting

**Default:** PANEL_HORIZONTAL

**Procs:** create, get, set

**Objects:** Panel, Panel_item

**See Also:** 7.3.1

**PANEL_LINE_BREAK_ACTION**

At the end of each line, wrap to the next line at either the character (PANEL_WRAP_AT_CHAR) or word (PANEL_WRAP_AT_WORD) level.

**Argument:** Panel_setting

**Valid values:** [PANEL_WRAP_AT_CHAR, PANEL_WRAP_AT_WORD]

**Default:** value of “text.lineBreak” default, or PANEL_WRAP_AT_WORD

**Procs:** create, get, set

**Objects:** Panel_multiline_text_item

**See Also:** 7.17

**PANEL_LIST_CLIENT_DATA**

Sets or gets up to 32 bits of user-entered data (argument 2) from a row number (argument 1) on the list.

**Argument 1:** int

**Argument 2:** XvOpaque

**Default:** NULL

**Procs:** create, get, set

**Objects:** Panel_list_item

**See Also:** 7.11.4.1
PANEL_LIST_CLIENT_DATAS

Works much like its companion attribute, PANEL_LIST_CLIENT_DATA, except that it takes a NULL-terminated value list of client data as its value. Position in the value list determines the row to which the data will be associated.

Argument:  NULL-terminated list of Xv_opaque  
Default:  NULL  
Procs:  create, set  
Objects:  Panel_list_item  
See Also:  7.11.4.1

PANEL_LIST_DELETE

Deletes a row from the scrolling list. The list is adjusted automatically after the deletion.

Argument:  int (row number)  
Default:  None  
Procs:  create, set  
Objects:  Panel_list_item  
See Also:  7.11.2

PANEL_LIST_DELETE_INACTIVE_ROWS

Deletes all inactive rows from the list. Similar to the PANEL_LIST_DELETE_SELECTED_ROWS attribute.

Argument:  No argument  
Procs:  set  
Objects:  Panel_list_item  

PANEL_LIST_DELETE_ROWS

Deletes the specified number of rows, starting at the specified row. The first argument is the row number, and the second argument is the number of rows to delete.

Argument 1:  int  
Argument 2:  int  
Procs:  create, set  
Objects:  Panel_list_item  
See Also:  7.11.2

PANEL_LIST_DELETE_SELECTED_ROWS

Deletes all selected rows. Not valid in edit mode.

Argument:  No value  
Procs:  set  
Objects:  Panel_list_item  
See Also:  7.11.2

PANEL_LIST_DISPLAY_ROWS

Sets the number of rows in a list that will be displayed. This may not be set to less than three for a FILE_LIST.

Argument:  int  
Default:  The number of items in the xv_create() call, up to a maximum of 5. For a FILE_LIST the default is 10.  
Procs:  create, get, set  
Objects:  Panel_list_item, File_list  
See Also:  7.11.1
**PANEL_LIST_DO_DBL_CLICK**

Tells PANEL_LIST to interpret two select events that occur within the timeout value as a double-click instead of as a second select or a deselect (depending on the current mode of the list). The timeout value is specified with OpenWindows.MulticlickTimeout. If true, the PANEL_LIST will deliver a new op called PANEL_LIST_OP_DBL_CLICK instead of the normal PANEL_LIST_OP_SELECT or PANEL_LIST_OP_DESELECT.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Panel_list_item

**PANEL_LIST_EXTENSION_DATA**

Same as PANEL_LIST_CLIENT_DATA, except that it is reserved for package implementors. This is used by the FILE_LIST package.

**PANEL_LIST_EXTENSION_DATAS**

Same as PANEL_LIST_CLIENT_DATAS, except that it is reserved for package implementors. This is used by the FILE_LIST package.

**PANEL_LIST_FIRST_SELECTED**

Returns the row number of the first selected row. If no row is selected, it returns -1.

Return Type: int
Argument: No argument
Procs: get
Objects: Panel_list_item
See Also: 7.11.3

**PANEL_LIST_FONT**

Set the specified row (argument 1) to the specified font (argument 2). Note that the font specification using PANEL_LIST_FONT should follow the creation of the row (using PANEL_LIST_STRINGS).

Argument 1: int (row number)
Argument 2: Xv_Font
Default: Use panel font
Procs: create, get, set
Objects: Panel_list_item
See Also: 7.11.1

**PANEL_LIST_FONTS**

Set the first \( n \) rows to the specified fonts. Note that fonts specified using PANEL_LIST_FONT should follow the creation of the rows (using PANEL_LIST_STRINGS).

Argument: NULL-terminated list of Xv_Font
Default: Use panel font
Procs: create, set
Objects: Panel_list_item
See Also: 7.11.1
PANEL_LIST_GLYPH
Takes a row number (argument 1) and Server_image (argument 2) to let you assign a glyph or icon to a row. The height of the glyph may not exceed the height of the scrolling list row. Also see PANEL_LIST_ROW_HEIGHT.

Return Type: Server_image
Argument 1: int
Argument 2: Server_image
Default: NULL
Procs: create, get, set
Objects: Panel_list_item
See Also: 7.11.1

PANEL_LIST_GLYPHS
Works the same as its companion attribute, PANEL_LIST_GLYPH, except that it takes a NULL-terminated value list as its value. The height of the glyph may not exceed the height of the scrolling list row. Also see PANEL_LIST_ROW_HEIGHT.

Argument: NULL-terminated list of Server_image
Default: NULL
Procs: create, set
Objects: Panel_list_item
See Also: 7.11.1

PANEL_LIST_INACTIVE
Grays out a row in a PANEL_LIST. Note that a row that is inactive cannot be selected at the same time. Also, mouseless navigation is not effected by the inactive state of the individual rows.

Argument 1: int
Argument 2: Bool
Default: FALSE
Procs: create, get, set
Objects: Panel_list_item

PANEL_LIST_INSERT
Inserts a list item at a specified row number. This attribute allocates space, attaches a row number to the list, and inserts an empty string. Clients must set PANEL_LIST_GLYPH and/or PANEL_LIST_STRING at this row number to set the glyph item and/or string.

Argument: int
Procs: create, set
Objects: Panel_list_item
See Also: 7.11.2

PANEL_LIST_INSERT_DUPLICATE
Allow (TRUE) or disallow (FALSE) duplicate strings to be inserted into the scrolling list.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Panel_list_item
See Also: 7.11.2
PANEL_LIST_INSERT_GLYPHS
Insert the specified glyphs into the scrolling list before the specified row. The first argument is a row number, and the second argument is a pointer to a NULL-terminated array of server images.

Argument 1: int
Argument 2: NULL-terminated list of Server_image
Procs: create, set
Objects: Panel_list_item
See Also: 7.11.2

PANEL_LIST_INSERT_STRINGS
Insert the specified strings into the scrolling list before the specified row. The first argument is a row number, and the second argument is a pointer to a NULL-terminated array of strings. The strings are copied when this attribute is set.

Argument 1: int
Argument 2: NULL-terminated list of char *
Procs: create, set
Objects: Panel_list_item
See Also: 7.11.2

PANEL_LIST_MASK_GLYPH
Tells the PANEL_LIST to use the given Server_image as a clip mask for the corresponding PANEL_LIST_GLYPH. The Server_image supplied must be of depth 1.

Argument: int
Argument: Server_image
Default: XV_NULL
Procs: create, get, set
Objects: Panel_list

PANEL_LIST_MASK_GLYPHS
Like PANEL_LIST_MASK_GLYPH, but takes a NULL terminated list of glyphs rather than a row and a single handle.

Argument: NULL-terminated list of Server_image
Procs: create, set
Objects: Panel_list

PANEL_LIST_MODE
Sets or gets the mode of the scrolling list. Setting the mode to PANEL_LIST_READ, when in edit mode is equivalent to selecting the “End Editing” menu item. Setting the mode to PANEL_LIST_EDIT when in read mode is equivalent to selecting the “Edit List” menu item.

Argument: Panel_list_mode
Valid Values: {PANEL_LIST_READ, PANEL_LIST_EDIT}
Default: PANEL_LIST_READ
Procs: get, set
Objects: Panel_list_item
See Also: 7.11.5
PANEL_LIST_NEXT_SELECTED

Returns the row number of the first selected row following the row specified. If no row after the row specified is selected, returns -1.

Argument: int
Procs: get
Objects: Panel_list_item
See Also: 7.11.3

PANEL_LIST_NROWS

Gets the total number of rows in the scrolling list.

Return Type: int
Procs: get
Objects: Panel_list_item

PANEL_LIST_ROW_HEIGHT

Specifies the height of each row in the scrolling list.

Argument: int
Default: Height of panel’s font
Procs: create, get
Objects: Panel_list_item
See Also: 7.11.1

PANEL_LIST_ROW_VALUES

Improved performance method of getting/setting row values into the PANEL_LIST. Takes the row number, a pointer to a Panel_list_row_values array, and a count of how many rows in the array. Panel_list_row_values defined shown below.

Argument 1: int
Argument 2: struct *
Argument 3: int
Procs: create, get, set
Objects: Panel_list

typedef struct {
    char * string;
    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;

On xv_get, the arguments remain the same, the struct passed in gets filled in by the PANEL_LIST package. The return value is the number of rows that were successfully filled in.
PANEL_LIST_SCROLLBAR
Returns the scrollbar attached to a PANEL_LIST panel item.

Return Type: Scrollbar
Procs: get
Objects: Panel_list_item

PANEL_LIST_SELECT
Takes two values: a row number (argument 1) and a Bool (argument 2) that lets you select (TRUE) or
deselect (FALSE) the specified row. If the scrolling list is not hidden (XV_SHOW is TRUE) then the
specified row will be made visible, which may entail scrolling the scrolling list. To disable this scrol-
loring, set XV_SHOW to FALSE before setting PANEL_LIST_SELECT and then reset XV_SHOW to TRUE af-
terwards.

Argument 1: int
Argument 2: Bool
Default: FALSE
Procs: set, create
Objects: Panel_list_item
See Also: 7.11.3

PANEL_LIST_SELECTED
Returns whether the specified row number is selected (TRUE) or not (FALSE).

Argument: int
Return Type: Bool
Procs: get
Objects: Panel_list_item

PANEL_LIST_SORT
Sort the list in forward (PANEL_FORWARD) or reverse (PANEL_REVERSE) alphabetical order. The
FILE_LIST package consumes this attribute. Use FILE_LIST_COMPARE_FUNC in its place.

Argument: Panel_setting
Valid values: PANEL_FORWARD or PANEL_REVERSE
Procs: set
Objects: Panel_list_item

PANEL_LIST_STRING
Specifies the string (argument 2) of a specified row (argument 1). xv_get returns the pointer to the
character string assigned to the row. The string is copied when this attribute is set.

Return Type: char *
Argument 1: int (row number)
Argument 2: char *
Default: Empty string
Procs: create, get, set
Objects: Panel_list_item
See Also: 7.11.1
**PANEL_LIST_STRINGS**

Works the same as its companion attribute, PANEL_LIST_STRING, except that it takes a NULL-terminated list of strings as its value. The strings are copied when this attribute is set.

- **Argument:** NULL-terminated list of char *
- **Default:** Empty string
- **Procs:** create, set
- **Objects:** Panel_list_item
- **See Also:** 7.11.1

**PANEL_LIST_TITLE**

The title of the scrolling list. PANEL_LIST_TITLE makes a copy of the string. When the scrolling list is removed, the package will free the title string.

- **Argument:** char *
- **Default:** No title
- **Procs:** create, get, set
- **Objects:** Panel_list_item
- **See Also:** 7.11.1

**PANEL_LIST_WIDTH**

Specifies the width of the scrolling list: -1 extends the scrolling list to the right edge of the panel. A value of 0 sets the width slightly wider than the widest row, and other values specify the width of the scrolling list in pixels. To get the width of a PANEL_LIST item's rectangle, use XV_WIDTH. To get the width of the label rectangle, use PANEL_LABEL_WIDTH.

- **Argument:** int
- **Default:** 0
- **Procs:** create, get, set
- **Objects:** Panel_list_item
- **See Also:** 7.11.1

**PANEL_MASK_CHAR**

Specifies the character used to mask type-in characters. Use the space character for no character echo (caret does not advance). Use the NULL character to disable masking.

- **Argument:** char
- **Default:** NULL
- **Procs:** create, set, get
- **Objects:** Panel_text_item, Panel_numeric_text_item
- **See Also:** 7.15

**PANEL_MAX_TICK_STRING**

String which appears underneath the maximum tick mark on horizontal sliders or gauges, or to the right of the maximum tick mark on vertical sliders or gauges. If PANEL_TICKS is 0, PANEL_MAX_TICK_STRING is ignored. The width of the slider or gauge may be adjusted to insure that there is enough space to accommodate both the minimum and maximum tick strings. The string is copied when this attribute is set.

- **Argument:** char *
- **Default:** No maximum tick string
- **Procs:** create, get, set
- **Objects:** Panel_slider_item, Panel_gauge_item
- **See Also:** 7.13

---

**XView Attributes**

115
**PANEL_MAX_VALUE**

Specifies the maximum value of the slider, gauge, or numeric text item.

- **Argument:** int
- **Default:** 100
- **Procs:** create, get, set
- **Objects:** Panel_numeric_text_item, Panel_gauge_item, Panel_slider_item
- **See Also:** 7.13

**PANEL_MAX_VALUE_STRING**

Maximum value string 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. The string is copied when this attribute is set.

- **Argument:** char *
- **Default:** No maximum value string
- **Procs:** create, get, set
- **Objects:** Panel_slider_item
- **See Also:** 7.13

**PANEL_MIN_TICK_STRING**

String which appears underneath the minimum tick mark on horizontal sliders or gauges, or to the right of the minimum tick mark on vertical sliders or gauges. If PANEL_TICKS is 0, PANEL_MIN_TICK_STRING is ignored. The width of the slider or gauge may be adjusted to insure that there is enough space to accommodate both the minimum and maximum tick strings. The string is copied when this attribute is set.

- **Argument:** char *
- **Default:** No minimum tick string
- **Procs:** create, get, set
- **Objects:** Panel_slider_item, Panel_gauge_item
- **See Also:** 7.13

**PANEL_MIN_VALUE**

Specifies the minimum value of the slider, gauge, or numeric text item.

- **Argument:** int
- **Default:** 0
- **Procs:** create, get, set
- **Objects:** Panel_slider_item, Panel_numeric_text_item, Panel_gauge_item
- **See Also:** 7.13

**PANEL_MIN_VALUE_STRING**

Minimum value 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. The string is copied when this attribute is set.

- **Argument:** char *
- **Default:** No minimum value string
- **Procs:** create, get, set
- **Objects:** Panel_slider_item
- **See Also:** 7.13
**PANEL_NCHOICES**

Returns the number of choices available in a choice or toggle item.

Return Type: `int`
Procs: `get`
Objects: `Panel_choice_item`
See Also: 7.10

**PANEL_NEXT_COL**

This attribute is used when the panel layout is `PANEL_VERTICAL`. It specifies that the item is to start a new column and specifies the amount of white space (in pixels) between the last column and the next (new) column. If you specify −1, the default gap (PANEL_ITEM_X_GAP) is used.

Argument: `Panel_item`
Default: None
Procs: `create`
Objects: `Panel_item`
See Also: 7.3.1

**PANEL_NEXT_ITEM**

Gets the handle of the next item on a panel.

Argument: `Panel_item`
Procs: `get`
Objects: `Panel_item`
Usage:

```c
next_item = xv_get(this_item, PANEL_NEXT_ITEM);
```

See Also: 7.7

**PANEL_NEXT_ROW**

This attribute is used when the panel layout is `PANEL_HORIZONTAL`. It specifies that the item is to start a new row and specifies the amount of white space (in pixels) between the last row and the next (new) row. If you specify −1, the default gap (PANEL_ITEM_Y_GAP) is used.

Argument: `Panel_item`
Default: None
Procs: `create`
Objects: `Panel_item`
See Also: 7.3.1

**PANEL_NO_REDISPLAY_ITEM**

This flag is useful if you intend to call `xv_super_set_avlist()` at the beginning of the set routine. Setting this attribute to `TRUE` prevents the parent panel from redisplaying while the attributes are being set. This attribute should be reset to `FALSE` after the call to `xv_super_set_avlist()`.

Warning: This attribute should only be used from within a panel item extension.

Argument: `Bool`
Procs: `get,set`
Objects: `Panel`
See Also: 25.11.6
**PANEL_NOTIFY_LEVEL**

Specifies when to call the notify function. The valid values are PANEL_ALL, PANEL_NONE, PANEL_SPECIFIED, and PANEL_NON_PRINTABLE. For sliders, any setting other than PANEL_ALL results in notification only on SELECT-up.

- **Argument:** Panel_setting
- **Valid Values:** [PANEL_ALL, PANEL_NONE, PANEL_SPECIFIED, PANEL_NON_PRINTABLE]
- **Default:** PANEL_SPECIFIED
- **Procs:** create, get, set
- **Objects:** Panel_numeric_text_item, Panel_multiline_text_item, Panel_slider_item, Panel_text_item
- **See Also:** 7.13.2

**PANEL_NOTIFY_PROC**

Procedure to call when a panel item is activated.

- **Argument:** Varies—see callbacks below
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Panel_item

**Callback:** (For Button Items)

```c
void
notify_proc(item, event)
    Panel_item item;
    Event *event;
```

Client-specified callback routine invoked when the user activates the button (accepts preview).

**Callback:** (For Message Items)

```c
void
notify_proc(item, event)
    Panel_item item;
    Event *event;
```

Client-specified callback routine invoked when the user activates the message (accepts preview).

**Callback:** (For Text Items, Numeric Text Items, and Multiline Text Items)

```c
Panel_setting
notify_proc(item, event)
    Panel_item item;
    Event *event;
```

Client-specified callback routine invoked when the user activates the text (accepts preview). For text, numeric text, and multiline text items, the return value type is Panel_setting and is one of PANEL_INSERT, PANEL_NEXT, PANEL_NONE, or PANEL_PREVIOUS.
Callback: (For Exclusive Choice Items and Slider Items)

```c
void
notify_proc(item, value, event)
    Panel_item item;
    int value;
    Event *event;
```

Client-specified callback routine invoked when the user activates the exclusive choice or slider (accepts preview).

Callback: (For Non-exclusive Choice Items, toggles)

```c
void
notify_proc(item, value, event)
    Panel_item item;
    unsigned int value;
    Event *event;
```

Client-specified callback routine invoked when the user activates the non-exclusive choice (accepts preview). The value parameter is of type unsigned int because it represents a mask of choices that are selected. For example, if the first and third items are selected, then the first and third bits in the value parameter are on—this value happens to be five.

Callback: (For List Items)

```c
int
notify_proc(item, string, client_data, op, event, row)
    Panel_item item;
    char *string;
    Xv_opaque client_data;
    Panel_list_op op;
    Event *event;
    int row;
```

Client-specified callback routine invoked when the user activates the list item (accepts preview).

string is the string associated with the row.
client_data is the client data associated with the row.
op indicates a select, validate, or delete operation. Of these, only PANEL_LIST_OP_VALIDATE requires a return value.
row indicates the row in the scrolling list. Returns XV_OK to validate a change or XV_ERROR to invalidate a change.

Callback: (For Drop Target Item)

```c
int
notify_proc(item, event)
    Panel_item item;
    Event *event;
```

A return value of XV_OK indicates that XView should call dnd_done(). A return value of XV_ERROR indicates that XView should not call dnd_done().

---

**XView Attributes**

119
Callback: (For Path Name Items)

```c
Panel_setting
notify_proc(item, event, stat_buf)
Path_name item;
Event * event;
struct stat * stat_buf;
```

Same as for PANEL_TEXT, except that the PATH_NAME package passes in the pointer to a stat structure as well as the item and event handles.

Callback: (For a File_list)

```c
int notify_proc( item, dir, file, client_data, op, event, row )
File_list item;
char *dir;
char *file;
Xv_opaque client_data;
Panel_list_op op;
Event *event;
int row;
```

Note: the client_data field is added either by the client explicitly setting PANEL_LIST_CLIENT_DATA(S) or by returning the value from the FILE_LIST_FILTER_FUNC.

The op PANEL_LIST_OP_DBL_CLICK, supports the PANEL_LIST package for applications that want to detect a double-click condition. This behavior is selected using the attribute PANEL_LIST_DO_DBL_CLICK which defaults to FALSE for the PANEL_LIST package (for compatibility reasons) but defaults to TRUE for the FILE_LIST package.

The FILE_LIST package intercepts the PANEL_LIST_OP_DBL_CLICK condition in cases where the user double-clicks on a directory name. The client only sees the double-click op if it occurred on a file that he may wish to open or save to. The client can still catch the directory change by installing a function via the FILE_LIST_CHANGE_DIR_FUNC attribute.

See Also: 7.10.5

**PANEL_NOTIFY_STATUS**

If the panel item is part of an unpinned command frame, then the window is dismissed if PANEL_NOTIFY_STATUS is XV_OK. If PANEL_NOTIFY_STATUS is set to XV_ERROR, then the window is not dismissed. You only xv_get() or xv_set() this attribute from within a notify procedure. When a notify procedure exits, for an unpinned command frame, XView internally uses the value of this attribute to determine whether or not the command frame is dismissed. XView sets the value of this attribute to XV_OK before calling the notify procedure. For the PATH_NAME package, see the following PANEL_NOTIFY_STATUS explanation.

**Argument:** int

**Valid Values:** {XV_OK, XV_ERROR}

**Default:** XV_OK

**Procs:** get, set

**Objects:** Panel_item

**See Also:** 7.9.1
**PANEL_NOTIFY_STATUS**

This attribute is overloaded by the Path Name package to represent whether or not the PANEL_NOTIFY_PROC succeeded in validating the the last directory it was called with. The return value from the notify procedure is of type Panel_setting and does not exactly match the same condition. Expected values are XV_OK or XV_ERROR in the case of an invalid path name. For a PATH_ITEM, see the preceding PANEL_NOTIFY_STATUS explanation.

Argument: int
Default: XV_OK
Procs: create, get, set

**PANEL_NOTIFY_STRING**

String of characters that triggers notification when one of the characters is typed in a text item. Applies only when PANEL_NOTIFY_LEVEL is PANEL_SPECIFIED. The string is copied when this attribute is set.

Argument: char *
Default: \n\r\t (i.e., newline, carriage return, and tab)
Procs: create, get, set
Objects: Panel_numeric_text_item, Panel_text_item, Panel_multiline_text_item
See Also: 7.15.3

**PANEL_OPS_VECTOR**

The panel operations vector, which is the method by which the panel package dispatches events to panel items. For more information, see Section 25.11, “The Wizzy Package” in the XView Programming Manual.

Argument: Panel_ops *
Procs: get, set
Objects: Panel_item

**PANEL_PAINT**

Controls the panel item’s painting behavior in xv_set calls. The specified value applies only to the xv_set call. As the xv_set call exits, it restores PANEL_PAINT to its previous value. Possible values are PANEL_CLEAR, PANEL_NO_CLEAR, or PANEL_NONE. PANEL_PAINT operates on Panel and Panel_item objects. The default for Panel item objects is the value of PANEL_PAINT for the Panel to which the Panel item is attached. This attribute is used strictly for performance enhancements and, normally, is not required.

Argument: Panel_setting
Default: PANEL_CLEAR for panel objects, the panel’s PANEL_PAINT for panel item objects
Procs: create, set
Objects: Panel, Panel_item
See Also: 7.19.3

**PANEL_PRIMARY_FOCUS_ITEM**

The current or last panel item that is or was a primary (First-Class) focus client.

Warning: This attribute should only be used from within a panel item extension.

Argument: Panel_item
Procs: get, set
Objects: Panel
Usage:

```c
if (xv_get(panel_public, PANEL_PRIMARY_FOCUS_ITEM) == item_public)
    xv_set(panel_public, PANEL_PRIMARY_FOCUS_ITEM, NULL, NULL);
```
This example should be executed when an item’s remove procedure is called, if the item
is a primary focus client.

**PANEL_READ_ONLY**

If **TRUE**, editing is disabled; if **FALSE**, editing is enabled. Note that *xv_set* does not apply to scrol-
ing lists. **PANEL_READ_ONLY** mode defaults to **TRUE** for the **FILE_LIST** package as opposed to
**FALSE** for the **PANEL_LIST** package.

- **Argument:** Bool
- **Default:** **FALSE**
- **Procs:** create, get, set
- **Objects:** Panel_list_item, Panel_multiline_text_item, Panel_numeric_text_item, Panel_text_item, Panel_slider_item, File_list

**PANEL_REPAINT_PROC**

Specifies the name of the client-supplied panel background repaint procedure.

- **Argument:** void (\*panel_repaint_proc) ()
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Panel
- **Callback:**

```
void
panel_repaint_proc(panel, pw, repaint_area)
    Panel  panel;
    Xv_Window  pw;  /* paint window */
    Rectlist  *repaint_area;
```

**See Also:** 7.19.2

**PANEL_SHOW_RANGE**

If **TRUE**, shows the minimum and maximum slider or gauge values.

- **Argument:** Bool
- **Default:** **TRUE**
- **Procs:** create, get, set
- **Objects:** Panel_slider_item, Panel_gauge_item

**See Also:** 7.13

**PANEL_SHOW_VALUE**

If **TRUE**, shows current slider value.

- **Argument:** Bool
- **Default:** **TRUE**
- **Procs:** create, get, set
- **Objects:** Panel_slider_item

**See Also:** 7.13

**PANEL_SLIDER_END_BOXES**

Shows or hides the slider end boxes.

- **Argument:** Bool
- **Default:** **FALSE**
- **Procs:** create, get, set
- **Objects:** Panel_slider_item

**See Also:** 7.13
**PANEL_SLIDER_WIDTH**

Specifies the length of the slider bar in pixels. The length is set whether the slider is horizontally or vertically oriented.

- **Argument:** int
- **Default:** 100
- **Procs:** create, get, set
- **Objects:** Panel_slider_item
- **See Also:** 7.13

**PANEL_STATUS**

Returns the state of the panel. See `<xview/panel.h>` for more information.

- **Warning:** This attribute should only be used from within a panel item extension.

- **Argument:** Panel_status *
- **Procs:** get
- **Objects:** Panel

**PANEL_TEXT_SELECT_LINE**

Selects and highlights the entire contents of the text field.

- **Argument:** No value
- **Procs:** set
- **Objects:** Panel_text_item

**Usage:**

```
xv_set(ptext, PANEL_TEXT_SELECT_LINE, NULL);
```

- **See Also:** 7.15.2

**PANEL.Ticks**

Specifies the number of evenly-spaced tick marks to be displayed on slider or gauge panel items.

- **Argument:** int
- **Default:** 0
- **Procs:** create, get, set
- **Objects:** Panel_slider_item, Panel_gauge_item
- **See Also:** 7.13

**PANEL_TOGGLE_VALUE**

Specifies the value (argument 2) of a particular toggle choice (argument 1).

- **Argument 1:** int
- **Argument 2:** int
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Panel_choice_item

**PANEL.VALUE**

Indicates the current value of a panel item. Its type varies depending on the type of panel item specified. See Chapter 7, Panels, in the Xview Programming Manual for details.

- **Argument:** Varies with panel item type
- **Procs:** create, get, set
- **Objects:** Panel_item
- **See Also:** 7.6
PANEL_VALUE_DISPLAY_LENGTH
Maximum number of characters to display in a text string. Note that the length of the value display may not be less than the combined width of the left and right “more text” buttons. In 12-point font, this is four characters. This attribute is intended for use with fixed-width fonts. For variable-width fonts, use the attribute PANEL_VALUE_DISPLAY_WIDTH. For a Multiline Text Item, this specified the length of each row (line) in the Multiline Text Field, expressed in characters.

Argument: int
Default: 80 (for Slider, Text, and Numeric Text Items)
40 (for a Multiline Text Item)

Procs: create, get, set
Objects: Panel_slider_item, Panel_numeric_text_item, Panel_text_item,
         Panel_multiline_text_item

See Also: 7.17, 7.15

PANEL_VALUE_DISPLAY_WIDTH
The width, in pixels, of a text field’s value. For a Multiline Text Item, this is the width of each row (line) in the Multiline Text Field.

Argument: int
Default: 80 default character widths (for Text and Numeric Text Items).
          Width corresponding to value of PANEL_VALUE_DISPLAY_LENGTH (for a Multiline Text Item)

Procs: create, get, set
Objects: Panel_text_item, Panel_numeric_text_item,
         Panel_multiline_text_item

See Also: 7.17, 7.15

PANEL_VALUE_FONT
Specifies the font to use in the value portion of the Panel Item.
Warning: Use of this attribute may cause your application to violate the OPEN LOOK Graphical User Interface Functional Specification.

Argument: Xv_Font
Default: Panel’s font

Procs: create, get, set
Objects: Panelnumeric_text_item, Panel_slider_item, Panel_text_item

PANEL_VALUE_STORED_LENGTH
Maximum number of characters allowed in string value for a text item. For Multiline Text Items, if PANEL_VALUE_STORED_LENGTH is greater than PANEL_VALUE_DISPLAY_LENGTH times PANEL_DISPLAY_ROWS, then a scrollbar will be visible on the Multiline Text Field. (An outstanding bug in TEXTSW_MEMORY_MAXIMUM prevents PANEL_VALUE_STORED_LENGTH from being set to less than 1024; thus a Multiline Text Item will always have a scrollbar.)

Argument: int
Default: Same as TEXTSW_MEMORY_MAXIMUM default (for Multiline Text Item)
          80 (for a Text or Numeric Text Item)

Procs: create, get, set
Objects: Panel_numeric_text_item, Panel_text_item,
         Panel_multiline_text_item, Panel_list_item, Panel

See Also: 7.17, 7.15
**PANEL_VALUE_UNDERLINED**

Show or hide the text field value's underlining.

**Argument:** Bool
**Default:** TRUE
**Procs:** create, get, set
**Objects:** Panel_text_item, Panel_numeric_text_item

**PANEL_VALUE_X**

Specifies the x coordinate of the value portion of panel items that have separate label and value entities (e.g., text items).

**Argument:** int
**Default:** (See Chapter 7, Panels, in the XView Programming Manual.)
**Procs:** create, get, set
**Objects:** Panel_item
**See Also:** 7.4.3, PANEL_LABEL_X

**PANEL_VALUE_Y**

Specifies the y coordinate of the value portion of panel items that have separate label and value entities (e.g., text items).

**Argument:** int
**Default:** (See Chapter 7, Panels, in the XView Programming Manual.)
**Procs:** create, get, set
**Objects:** Panel_item
**See Also:** 7.4.3, PANEL_LABEL_Y

**PATH_IS_DIRECTORY**

Specifies if the PATH_NAME package should accept a path name as a directory or a full path to a file name. Changing this value from FALSE to TRUE has the side effect of setting the PATH_LAST_VALIDATED value to NULL if it is not a directory.

**Argument:** Bool
**Default:** FALSE
**Procs:** create, get, set
**Objects:** Path_name
**See Also:** Appendix D, Version 3.2 and the File Chooser, of XView Programming Manual.

**PATH_LAST_VALIDATED**

PANEL_VALUE returns the current contents of the Textfield, PATH_LAST_VALIDATED returns the last path name to pass validation in the Textfield. This means that you are always guaranteed of receiving a valid path name with this attribute. Note that this is always the expanded version of the path name, whereas PANEL_VALUE will return the current contents of the field.

**Warning:** if no path name has passed validation, this will return a NULL. Also, setting PATH_IS_DIRECTORY will cause this value to become NULL.

**Argument:** char *
**Default:** NULL
**Procs:** get
**Objects:** Path_name
**See Also:** Appendix D, Version 3.2 and the File Chooser, of XView Programming Manual.
PATH_RELATIVE_TO

Specifies an absolute path that any relative path input will be appended to in order to complete the path name. This means that if someone tries to validate the string `bin`, the PATH_NAME package will append `bin` to this string to determine what the actual path name should be validated as.

Argument: char *
Default: NULL
Procs: create, get, set
Objects: Path_name

PATH_USE_FRAME

Specifies if the PATH_NAME package is allowed to put error messages into its parent Frame’s footer.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Path_name

SCREEN_NUMBER

Specifies the number of the screen associated with object.

Argument: int
Default: 0
Procs: get
Objects: Screen
See Also: 15.2

SCREEN_SERVER

Specifies the server associated with this screen. By default, its value is the server created by opening the display specified by the value of the DISPLAY environment variable.

Return Type: Xv_Server
Procs: get
Objects: Screen
See Also: 15.3.3

SCROLLBAR_COMPUTE_SCROLL_PROC

Converts physical scrollbar info to client object info. The default_compute_scroll_proc can be called to perform the functionality that the scrollbar package normally would. If a normalize procedure is not set, then the offset becomes the viewstart (after bounds checking) and the scrollbar package will scroll to this offset into the object.

Default: scrollbar_default_compute_scroll_proc
Procs: create, get, set
Objects: Scrollbar
Callback:

```c
void scrollbar_default_compute_scroll_proc(
    scrollbar_public, pos, available_cable,
    motion, offset, object_length)
```

Scrollbar scrollbar
Scrollbar scrollbar
int int
int int
Scroll_motion Scroll_motion
unsigned long *offset;
unsigned long *object_length;

Computes default scrolling based on the scroll position. pos is the position from the start
of the cable event. length is the length of the cable. offset and object_length are output parameters. The offset is given in client units to scroll by. The object length is in client units.

**SCROLLBAR_DIRECTION**
Sets orientation of the scrollbar as SCROLLBAR_VERTICAL or SCROLLBAR_HORIZONTAL.

- **Argument:** Scrollbar_setting
- **Valid Values:** [SCROLLBAR_VERTICAL, SCROLLBAR_HORIZONTAL]
- **Default:** None
- **Procs:** create, get
- **Objects:** Scrollbar

**SCROLLBAR_LAST_VIEW_START**
Specifies the offset of the view into the object prior to the last scroll.

- **Argument:** int
- **Default:** 0
- **Procs:** get
- **Objects:** Scrollbar

**SCROLLBAR_MENU**
Specifies a pointer to the scrollbar’s menu. Clients can add items to the default menu but cannot re-
move items from it.

- **Argument:** Menu
- **Default:** Scrollbar creates a default menu
- **Procs:** get
- **Objects:** Scrollbar

**SCROLLBAR_MOTION**
Provides the scrolling motion that results in a scrollbar_request event.

- **Argument:** enum
- **Procs:** get
- **Return Values:**
  - ABSOLUTE
  - POINT_TO_MIN (from here_to_top on menu)
  - PAGE_FORWARD
SCROLLBAR_NORMALIZE_PROC
Specifies a procedure that does normalization (sets the viewable area of the object that the scrollbar
scrolls). This function should return vstart. It takes the offset given by the compute_proc and
adjusts it. The scrollbar package will then scroll to this offset into the object. The scrollbar package
will call the scrollbar_compute_scroll_proc and the scrollbar_normalize_proc in
that order whenever any scrolling is done.
Default: NULL
Procs: create, get, set
Objects: Scrollbar
Callback:
    scrollbar_normalize_proc(sb, voffset, motion, &vstart)

SCROLLBAR_NOTIFY_CLIENT
Used by the Notifier. Indicates the client that is notified when the scrollbar is scrolled. See Chapter
Argument: Xv_opaque
Default: Subwindow scrollbar is attached to contents
Procs: create, get, set
Objects: Scrollbar
See Also: 10.4.1

SCROLLBAR_OBJECT_LENGTH
Specifies the length of the scrollable object in scrollbar units. Value must be greater than or equal to
zero.
Argument: int
Default: 0
Procs: create, get, set
Objects: Scrollbar
See Also: 10.2

SCROLLBAR_PAGE_LENGTH
Specifies the length of a page in scrollbar units for page-scrolling purposes.
Argument: int
Default: 0
Procs: create, get, set
Objects: Scrollbar
See Also: 10.2
SCROLLBAR_PIECES_PER_UNIT
Specifies the number of pixels constituting a scrollbar unit. For example, when scrolling a list of icons, each unit might be 64 pixels.
Argument: int
Default: 1
Procs: create, get, set
Objects: Scrollbar
See Also: 10.2

SCROLLBAR_SPLITTABLE
Indicates whether the object that contains the scrollbar is splittable.
Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Scrollbar
See Also: 5.6.1

SCROLLBAR_VIEW_LENGTH
Specifies the length of the viewing window in scrollbar units.
Argument: int
Default: 0
Procs: create, get, set
Objects: Scrollbar
See Also: 10.2

SCROLLBAR_VIEW_START
Specifies the current offset into the scrollbar object in scrollbar units. The value must be greater than or equal to zero.
Argument: int
Default: 0
Procs: create, get, set
Objects: Scrollbar
See Also: 10.2

SEL_APPEND_TYPE_NAMES
Same as SEL_TYPE_NAMES except that the new list is appended to the previously set type list.
Argument: List of string (STRING, NULL)
Default: N/A
Procs: create, set
Objects: Selection_requestor
Usage:

```
Selection_requestor sel;
sel = xv_create(panel, SELECTION_REQUESTOR,
                SEL_REPLY_PROC, SelectionReplyProc,
                NULL);
    xv_set(sel, SEL_APPEND_TYPE_NAMES, "TARGETS", NULL, NULL);
```
See Also: 18.2.3.1, SEL_TYPE_NAMES
**SEL_APPEND_TYPES**

Same as SEL_TYPES except that the new list is appended to the previously set type list.

Argument: List of Atoms (XA_STRING, NULL)
Default: N/A
Procs: create, set
Objects: Selection_requestor
Usage:

```c
Selection_requestor sel;
sel = xv_create(panel, SELECTION_REQUESTOR,
    SEL_REPLY_PROC, SelectionReplyProc, NULL);
xv_set(sel, SEL_APPEND_TYPES, XA_STRING, NULL, NULL);
```

See Also: 18.2.3.1, SEL_TYPES

**SEL_CONVERT_PROC**

Specifies the procedure that is to be called whenever a client requests the current value of the selection.

Argument: int (*convert_proc)()
Default: sel_convert_proc
Procs: create, get, set
Objects: Selection_owner
Callback:

```c
int
convert_proc( sel, replyType, replyBuff, length, format )
    Selection_owner sel;
    Atom *replyType;
    Xv_opaque *replyBuff;
    unsigned long *length;
    int *format;

    sel specifies the selection owner.
    replyType specifies the type of the selection that has been requested. It should be explicitly set to an atom which describes the converted type of the selection (for example, TEXT might have the type XA_STRING).
    replyBuff is a pointer to a buffer address which contains the converted data.
    length specifies a pointer to the number of elements in replyBuff. The size of an element is defined by format. The convert procedure is called with length set to the maximum allowed buffer size.
    format specifies a pointer to the data format. Valid values are 8, 12, or 32 for 8-bit, 16-bit, or 32-bit quantities, respectively.
```

See Also: 18.2.4

**SEL_COPY**

SEL_COPY indicates whether or not to make a copy of the SEL_DATA data. If set to FALSE it is up to the client to maintain the data.

Argument: Bool
Default: TRUE
Procs: create, set, get
Objects: Selection_item
See Also: 18.3.1
**SEL_DATA**

Used to initiate a blocking selection request. The arguments to this attribute are a pointer to a long which will be set to the number of elements in the returned buffer and a pointer to an integer which will be set to the data format. `xv_get()` returns a pointer to the selection data. Clients should free the returned buffer. If the requestor client has not registered a `reply_proc()` with the selection package and is requesting for `MULTIPLE` or `INCR`, the call will return with `length` set to `SEL_ERROR` and format set to zero. If the requestor client has registered a `reply_proc()` with the selection package and requesting for `MULTIPLE` or `INCR`, the package will call the client’s `reply_proc()` with the converted data. `xv_get()` returns after the transaction has completed with `length` set to `XV_OK` and format set to the returned data format. If the request fails the `xv_get()` will return with `length` set to `SEL_ERROR` and format set to zero. For a `Selection_item` object, this attribute specifies a pointer to the selection data.

**Argument:** XvOpaque

**Default:** N/A

**Procs:** get for `Selection_requestor` objects
get and set for `Selection_item` objects

**Objects:** `Selection_requestor`, `Selection_item`

**Usage:**

```c
data = (char *) xv_get(sel_requestor, SEL_DATA,
                        &length, &format);
```

**See Also:** 18.3.1, 18.2.3.4

**SEL_DONE_PROC**

Specifies the procedure that is called after the requestor has received the selection or `NULL` if the owner is not interested in being called back.

**Argument:** void(*done_proc)()

**Default:** N/A

**Procs:** create, get, set

**Objects:** `Selection_owner`

**Callback:**

```c
void done_proc( sel, replyBuff, target )
{
    Selection_owner sel;
    XvOpaque replyBuff;
    Atom target;

    sel specifies the selection owner.
    replyBuff specifies the address which contains the converted data.
    target specifies the target type returned by the convert procedure.
}
```

**See Also:** 18.2.7

**SEL_FIRST_ITEM**

Returns a selection item.

**Argument:** Selection_item

**Default:** N/A

**Procs:** get

**Objects:** `Selection_owner`

**See Also:** 18.3.1, `SEL_NEXT_ITEM`
**SEL_FORMAT**

Specifies the data format.

Argument:  Xv_opaque  
Default:  8  
Procs:  create, get, set  
Objects:  Selection_item  
See Also:  18.3.1

**SEL_LENGTH**

Specifies the number of 8, 16, 32-bit elements contained in the reply.

Argument:  unsigned long  
Default:  Type-dependent  
Procs:  create, get, set  
Objects:  Selection_item  
See Also:  18.3.1

**SEL_LOSE_PROC**

Used to register a procedure that is called back whenever the selection owner loses the selection that it holds.

Argument:  void (*lose_proc) ()  
Default:  NULL  
Procs:  create, get, set  
Objects:  Selection_owner  
Callback:

```c
void
lose_proc( sel )
    Selection_owner sel;
```

sel specifies the selection owner.

See Also:  18.2.6

**SEL_NEXT_ITEM**

Returns a selection item.

Argument:  Selection_item  
Default:  N/A  
Procs:  get  
Objects:  Selection_owner  
See Also:  18.3.1, SEL_FIRST_ITEM

**SEL_OWN**

Setting SEL_OWN causes the selection to be acquired (TRUE) or lost (FALSE). Owning a selection without an external conversion procedure or any selection items will generate a NULL-data reply to any incoming requests.

Argument:  Bool  
Default:  FALSE  
Procs:  create, get, set  
Objects:  Selection_owner  
See Also:  18.2.4.2
**SEL_PROP_DATA**
Specifies the data associated with a property. Should be used in conjunction with SEL_TYPE_INDEX.

Argument: Xv_opaque
Default: NULL
Procs: create, set
Objects: Selection_requestor
See Also: 18.6, SEL_TYPE_INDEX

**SEL_PROP_FORMAT**
Specifies the format of the data associated with a property. Should be used in conjunction with SEL_TYPE_INDEX.

Argument: int
Default: N/A
Procs: create, set
Objects: Selection_requestor
See Also: 18.6, SEL_TYPE_INDEX

**SEL_PROP_INFO**
Returns the property data. This attribute should be used from a conversion procedure. It returns the data that has been set on the selection notifier’s property by the requestor client.

Argument: Sel_prop_info *
Default: N/A
Procs: get
Objects: Selection_owner
See Also: 18.7, SEL_TYPE_INDEX

**SEL_PROP_LENGTH**
Specifies the length of the data associated with a property. Should be used in conjunction with SEL_TYPE_INDEX.

Argument: unsigned long
Default: N/A
Procs: create, set
Objects: Selection_requestor
See Also: 18.6, SEL_TYPE_INDEX

**SEL_PROP_TYPE**
Specifies the type of the data associated with a property.

Argument: Atom
Default: N/A
Procs: create, set
Objects: Selection_requestor
See Also: 18.6, SEL_PROP_INFO, SEL_TYPE_INDEX

**SEL_PROP_TYPE_NAME**
Specifies the type of the data associated with a property. It takes a string argument which is interned into an atom. Should be used in conjunction with SEL_TYPE_INDEX.

Argument: Atom
Default: N/A
Procs: create, set
Objects: Selection_requestor
See Also: 18.6, SEL_PROP_INFO, SEL_TYPE_INDEX
**SEL_RANK**

Used to set the rank of the selection. Pre-defined atoms are `XA_PRIMARY` and `XA_SECONDARY`. The client should set the rank to the atom representing the selection.

- **Argument:** Atom
- **Default:** `XA_PRIMARY`
- **Procs:** create, get, set
- **Objects:** Selection
- **See Also:** `SEL_RANK` and Chapter 12, *Interclient Communication*, in *Xlib Programming Manual*.  

**SEL_RANK_NAME**

Used to set the rank of the selection. The package will intern the selection atom using this atom name (a string).

- **Argument:** *char
- **Default:** `XA_PRIMARY`
- **Procs:** create, get, set
- **Objects:** Selection
- **See Also:** 18.2.2

**SEL_REPLY_PROC**

A procedure that is called when a response to a request comes in.

- **Argument:** `void (*reply_proc)()`
- **Default:** NULL
- **Procs:** create, get, set
- **Objects:** Selection_requestor
- **Callback:**

```c
void
reply_proc( sel_req, target, type, replyValue, length, format )
    Selection_requestor sel_req;
    Atom target;
    Atom type;
    Xv_opaque replyValue;
    unsigned long length;
    int format;
```

If the selection conversion fails, this routine is called with `replyValue` set to an error code and `length` set to `SEL_ERROR`. `sel_req` specifies the selection requestor. `target` specifies the target type set by the convert procedure. `type` specifies the type returned by the convert procedure. `replyValue` specifies the data content returned. `length` specifies the length of the data returned by the convert procedure. `format` specifies the format of the data returned by the convert procedure.

If the selection content is larger than the server’s maximum request size or if the selection owner has decided to transfer the selection data in increments, the selection package will send the data to the requestor in chunks.

**See Also:** 18.2.5
SEL_TIME
Specifies the time of the acquisition or the request attempt.

Argument: struct timeval *
Default: Last event time
Procs: create, get, set
Objects: Selection
See Also: 18.2.3

SEL_TIMEOUT_VALUE
Selection timeout value. This value indicates the number of seconds that a requestor or a selection owner waits for a response during the selection transfer.

Argument: unsigned int
Default: From the value of the resource Selection.Timeout
Procs: create, get, set
Objects: Selection
See Also: 18.2.3.3, 18.2.2

SEL_TYPE
Specifies an atom type that the client is requesting. For a selection item, specifies the type that the package will convert to.

Argument: Atom
Default: XA_STRING
Procs: create, get, set
Objects: Selection_item, Selection_requestor
See Also: 18.2.3.1, SEL_TYPES, SEL_TYPE_NAME, and Chapter 12, Interclient Communication, in Xlib Programming Manual.

SEL_TYPE_INDEX
Specifies an index to the SEL_TYPES or SEL_TYPE list. This attribute is also used to specify an index to the SEL_TYPE_NAME and SEL_TYPE_NAMES lists. This attribute is used in conjunction with SEL_PROP_DATA, SEL_PROP_LENGTH, SEL_PROP_FORMAT, SEL_PROP_TYPE, and SEL_PROP_TYPE_NAME to associate data with a property used by the requestor.

Argument: int
Default: N/A
Procs: create, set
Objects: Selection_requestor
Usage: (Multiple Request)

```c
xv_set( sel_req,
      SEL_TYPES, XA_STRING, INSERT_SELECTION, NULL,
      SEL_TYPE_INDEX, 1,
      SEL_PROP_DATA, dataPointer,
      SEL_PROP_LENGTH, 20,
      NULL );
```

Usage: (Single Request)

```c
xv_set( sel_req, SEL_TYPE, XA_STRING,
      SEL_TYPE_INDEX, 0,
      SEL_PROP_DATA, dataPointer,
      SEL_PROP_LENGTH, 20,
      NULL );
```

See Also: 18.6

XView Attributes 135
SEL_TYPE_NAME
Same as SEL_TYPE except that the argument is the name of the requested selection type instead of an atom. The package will intern the requested atom name. For a selection item, specifies the type name that the package will convert to.

Argument: *char
Default: "STRING"
Procs: create,get,set
Objects: Selection_requestor Selection_item
See Also: 18.2.3.1

SEL_TYPE_NAMES
Specifies a null-terminated list of atom type names that the client is requesting.

Argument: List of string
Default: "STRING, NULL"
Procs: create,get,set
Objects: Selection_requestor
See Also: 18.2.3.1

SEL_TYPES
Specifies a null-terminated list of atom types that the client is requesting. The effect will be as if a sequence of SelectionRequest events is delivered to the selection owner, one for each atom. This attribute will initiate a MULTIPLE request.

Argument: List of Atoms
Default: XA_STRING,NULL
Procs: create,get,set
Objects: Selection_requestor
See Also: 18.2.3.1

SELN_*
Appendix A presents all of the SELN_ attributes.

SERVER_ATOM
SERVER_ATOM is equivalent to XInternAtom() (with the only_if_exists flag set to FALSE) except that it caches the results on the server object so that subsequent requests for the same atom will not require a round-trip to the X server.

Return Type: Atom
Procs: get
Usage:

    Atom atom;
    atom = (Atom) xv_get(server_object,
                     SERVER_ATOM, "TIMESTAMP");

See Also: 15.3.3, SERVER_ATOM_NAME
SERVER_ATOM_NAME

SERVER_ATOM_NAME is equivalent to XGetAtomName() except that it caches the results on the server object so that subsequent requests will not require a roundtrip to the X server. The returned string is maintained by XView and should not be modified or freed. XView will free up all strings associated with atoms on that server when the server object is destroyed.

Return Type: char *
Procs: get
Usage:

```c
char *atom_name;
atom_name (char *)xv_get(server_object, atom);
```

See Also: 15.3.3, SERVER_ATOM

SERVER_EXTENSION_PROC

Specifies the procedure used to handle server extension events.

Argument: void (*extension_proc) ()
Default: NULL
Procs: create, get, set
Objects: Server
Callback:

```c
void extension_proc(display, xevent, window)
    Display *display;
    Event *xevent;
    Xv_window window;
```

See Also: 6.10

SERVER_EXTERNAL_XEVENT_MASK

This attribute, together with SERVER_EXTERNAL_XEVENT_PROC, allows a client to receive notification for X events destined for X windows which are not local to a client. For each non-local X window, the client can specify a different set of X events for which it needs notification. Additionally, an XView object handle is provided as an argument which is returned as a parameter during callback. For xv_create and xv_set this attribute takes these three arguments: an XID of a window, an event mask (see <X11/X.h> for details), and an XView object handle. SERVER_EXTERNAL_XEVENT_MASK for xv_get takes two arguments: an XID of a window, and an XView object handle and returns a mask.

Argument 1: unsigned long
Argument 2: unsigned long
Argument 3: Xv_opaque
Default: None
Procs: create, get, set
Objects: Server
Usage:

```c
xv_set (server, SERVER_EXTERNAL_XEVENT_MASK,
    RootWindow(dpy, 0),
    ButtonPressMask | PropertyChangeMask,
    frame,
    SERVER_EXTERNAL_XEVENT_PROC,
    root_event_proc, frame,
    NULL);
```

See Also: 6.11, SERVER_EXTERNAL_XEVENT_PROC

XView Attributes
SERVER_EXTERNAL_XEVENT_PROC

This attribute, together with attribute SERVER_EXTERNAL_XEVENT_MASK allows a client to receive notification for X events destined for X windows which are not local to the client. A client can register a separate callback procedure for each XView object handle. For xv_create and xv_set this attribute takes two arguments. For xv_get it takes one argument: an XView object handle and returns a function name. A NULL first argument temporarily disables callbacks.

Argument 1: void (*callback_pro)( )
Argument 2: Xv_opaque
Default: None
Procs: create,get,set
Objects: Server
Usage:

```c
xv_set (server,
    SERVER_EXTERNAL_XEVENT_MASK,
    RootWindow(dpy, 0),
    ButtonPressMask | PropertyChangeMask,
    frame,
    SERVER_EXTERNAL_XEVENT_PROC,
    root_event_proc, frame,
    NULL);
```

Callback:

```c
void
    callback_pro (server, display, xevent, handle)
    Xv_server    server;
    Display      *display;
    XEvent       *xevent;
    Xv_opaque    handle;
```

See Also: 6.11.SERVER_EXTERNAL_XEVENT_MASK

SERVER_IMAGE_BITMAP_FILE

Specifies a file containing the X11 bitmap from which the server image is created.

Argument: char *
Default: None
Procs: create
Objects: Server
See Also: 15.4.1

SERVER_IMAGE_BITS

Specifies the SunView pixrect image bits for the server image. Use SERVER_IMAGE_X_BITS for standard X11 bitmaps.

Argument: short *
Default: Uninitialized
Procs: create,get,set
Objects: Server_image
See Also: 15.4.1
SERVER_IMAGE_CMS
Specifies the colormap segment to be used in converting the server image data (which are logical colormap indices) into pixel values in the pixmap used to draw the server image.

Argument: cms
Default: The default cms for the application
Procs: create, get
Usage:

```c
Server_image image;
image = (Server_image)xv_create(NULL,
   SERVER_IMAGE,
   ...
   SERVER_IMAGE_CMS, cms,
   ...
   NULL);
```

See Also: 15.4.1

SERVER_IMAGE_COLORMAP
Specifies the name of the colormap segment to be used in converting the server image data (which are logical colormap indices) into pixel values in the pixmap used to draw the server image.

This is basically a backwards compatibility attribute, and where possible, SERVER_IMAGE_CMS should be used instead.

Argument: char *
Default: "xv_default_cms"
Procs: get, create
Usage:

```c
Cms cms;
Server_image image;

cms = (Cms)xv_create(NULL, CMS,
    ...
    CMS_NAME, "palette",
    ...
    NULL);

image = (Server_image)xv_create(NULL, SERVER_IMAGE,
    ...
    SERVER_IMAGE_COLORMAP, "palette",
    ...
    NULL);
```

SERVER_IMAGE_DEPTH
Specifies the bit plane depth of the server image.

Argument: int
Default: 1
Procs: create, get, set
Objects: Server_image
See Also: 15.4.1
SERVER_IMAGE_PIXMAP

Allows an existing pixmap to be associated with a server image. An xv_get using this attribute is equivalent to an xv_get of the XV_XID of the server image.

Argument: Pixmap
Default: None
Procs: create, get, set
Objects: Server_image
See Also: 15.4.1

SERVER_IMAGE_SAVE_PIXMAP

Allows the application to specify that the old pixmap must not be destroyed if a new pixmap is created as the result of changing any server image attributes or assigning a new pixmap directly. You should retain a handle to the old pixmap first.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Server_image
See Also: 15.4.1

SERVER_IMAGE_X_BITS

Specifies the bits to use in a server image. Bits are stored in an array of char.

Argument: char *
Default: None
Procs: create, set
Objects: Server_image
See Also: 15.4.1

SERVER_NTH_SCREEN

Specifies the screen with given number. Returns NULL if screen does not exist.

Argument 1: int
Argument 2: Screen
Default: N/A
Procs: create, get, set
Objects: Server
See Also: 15.2.1

SERVER_SYNC

Flushes the request buffer and waits for all events and errors to be processed by the server. If the argument is TRUE, then SERVER_SYNC discards all events on the input queue.

Argument: Bool
Default: N/A
Procs: set
Objects: Server

SERVER_SYNC_AND_PROCESS_EVENTS

Same as SERVER_SYNC, but this processes any events that arrive as a result of the XSync().

Argument: No value
Default: None
Procs: create, set
Objects: Server
See Also: 15.3.3
Appendix C, Textsw Action Attributes, describes the textsw action attributes. These attributes are only valid for use with a textsw notify procedure.

**TEXTSW_AGAIN_RECORDING**

If FALSE, changes to the textsw are not repeated when the user invokes AGAIN. Disabling reduces memory overhead.

- **Argument:** Bool
- **Default:** TRUE
- **Procs:** create, get, set
- **Objects:** Textsw

**TEXTSW_AUTO_INDENT**

If TRUE, automatically indents a new line to match the previous line.

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get, set
- **Objects:** Textsw

**TEXTSW_AUTO_SCROLL_BY**

Specifies the number of lines to scroll when type-in moves insert point below the view.

- **Argument:** int
- **Default:** 1
- **Procs:** create, get, set
- **Objects:** Textsw

**TEXTSW_BLINK_CARET**

Determines whether the caret blinks (for better performance don’t blink).

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get, set
- **Objects:** Textsw

**TEXTSW_BROWSING**

If TRUE, prevents editing of displayed text. If another file is loaded in, browsing stays on.

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get, set
- **Objects:** Textsw

**TEXTSW_CHECKPOINT_FREQUENCY**

Specifies the number of edits between checkpoints. Set to 0 to disable checkpointing.

- **Argument:** int
- **Default:** 0
- **Procs:** create, get, set
- **Objects:** Textsw
TEXTSW_CLIENT_DATA
   Specifies the pointer to arbitrary client data.
   Argument:    Xv_opaque
   Default:   NULL
   Procs:    create, get, set
   Objects:    Textsw

TEXTSW_CONFIRM_OVERWRITE
   Specifies confirmation of any request to write to an existing file.
   Argument:    Bool
   Default:   TRUE
   Procs:    create, get, set
   Objects:    Textsw

TEXTSW_CONTENTS
   Specifies the text for a subwindow. xv_get needs additional parameters (see Chapter 8, Text
   Argument:    char *
   Default:   NULL
   Procs:    create, get, set
   Objects:    Textsw
   See Also:  8.6.2

TEXTSW_CONTROL_CHARS_USE_FONT
   If FALSE, control characters always display as an up arrow followed by a character instead of what-
   ever glyph is in the current font.
   Argument:    Bool
   Default:   FALSE
   Procs:    create, get, set
   Objects:    Textsw

TEXTSW_DESTROY_VIEW
   The current view will be destroyed.
   Argument:    No value

TEXTSW_DISABLE_CD
   Stops textsw from changing current working directory and grays out associated items in the menu.
   Argument:    Bool
   Default:   FALSE
   Procs:    create, get, set
   Objects:    Textsw

TEXTSW_DISABLE_LOAD
   Prevents files from being loaded into the textsw and grays out the associated items in the menu.
   Argument:    Bool
   Default:   FALSE
   Procs:    create, get, set
   Objects:    Textsw
**TEXTSW_EDIT_COUNT**
Monotonically increments count of the number of edits made to the textsw.

Argument: int
Procs: get
Objects: Textsw

**TEXTSW_EXTRAS_CMD_MENU**
Returns the Extras submenu of the TEXT PANE menu for the textsw.

Return Type: Menu
Procs: get
Objects: Textsw

**TEXTSW_FILE**
For xv_create and xv_set, specifies the name of the file to load; for xv_get, returns the name of the file loaded or NULL if no file was loaded.

Argument: char *
Default: NULL
Procs: create, get, set
Objects: Textsw
See Also: 8.4.1

**TEXTSW_FILE_CONTENTS**
Initializes the text subwindow contents from a file, yet still edits the contents in memory as if specified using TEXTSW_FILE.

Argument: char *
Default: NULL
Procs: create, set
Objects: Textsw
See Also: 8.6.1

**TEXTSW_FIRST**
Specifies the zero-based index of first displayed character.

Argument: int
Procs: create, get, set
Objects: Textsw
See Also: 8.4.1

**TEXTSW_FIRST_LINE**
Specifies the zero-based index of first displayed line.

Return Type: int
Procs: get
Objects: Textsw
See Also: 8.7.2

**TEXTSW_FONT**
Specifies the font to use in a text subwindow.

Argument: Xv_Font
Procs: create, get, set
Objects: Textsw
See Also: 8.1
TEXTSW_HISTORY_LIMIT

Specifies the number of user action sequences that can be undone.

Argument: int
Default: 50
Procs: create, get, set
Objects: Textsw

TEXTSW_IGNORE_LIMIT

Specifies the number of edits textsw allows before vetoing destroy. Valid values are 0, which means the destroy will be vetoed if there have been any edits, and TEXTSW_INFINITY, which means the destroy will never be vetoed. Vetoing a destroy means a confirm notice is displayed when the textsw is about to be destroyed. This veto confirm message is displayed only if the ignore limit is set to 0. A textsw is destroyed when the textsw or its enclosing frame is the object of an xv_destroy() call, which occurs when the application is quit from the Window Manager menu.

Argument: int
Default: 0
Procs: create, get, set
Objects: Textsw

TEXTSW_INSERT_FROM_FILE

Inserts the contents of a file into a text subwindow at the current insertion point.

Argument: char *
Default: None
Procs: set, create
Objects: Textsw
See Also: 8.6.3

TEXTSW_INSERT_MAKES_VISIBLE

Controls whether insertion causes repositioning to make inserted text visible. Possible values are TEXTSW_ALWAYS or TEXTSW_IF_AUTO_SCROLL.

Argument: Textsw_enum
Default: TEXTSW_ALWAYS
Procs: create, get, set
Objects: Textsw
See Also: 8.7.4.2

TEXTSW_INSERTION_POINT

Specifies the index of the current insertion point.

Argument: Textsw_index
Default: None
Procs: create, get, set
Objects: Textsw
See Also: 8.4.3.1

TEXTSW_LENGTH

Specifies the length of the textsw’s contents.

Argument: int
Procs: get
Objects: Textsw
See Also: 8.3
**TEXTSW_LINE_BREAK_ACTION**

Determines how the textsw treats file lines that are too big to fit on one display line. TEXTSW_CLIP clips the line when it gets too long to fit; you don’t see any additional characters typed on the line when the line exceeds the width of the textsw. TEXTSW_WRAP_CHAR wraps the line around so the part of the line that exceeds the width of the textsw is displayed on the next line(s). Textsw does not insert a CR. It just displays it as if a CR were there. If this happens in the middle of the word, the textsw will display the line as if a CR were inserted in the middle of the word. TEXTSW_WRAP_WORD wraps at the word level, thus, it will not break a word up to wrap around. It will figure out where the word starts and display as if a CR were inserted before the word.

Argument: Textsw_enum
Valid Values: [TEXTSW_CLIP, TEXTSW_WRAP_AT_CHAR, TEXTSW_WRAP_AT_WORD]
Default: TEXTSW_WRAP_AT_WORD
Procs: create, get, set
Objects: Textsw
See Also: 8.7.1

**TEXTSW_LOWER_CONTEXT**

Specifies the minimum number of lines to maintain between insertion point and bottom of view. A value of −1 turns auto scrolling off.

Argument: int
Default: 0
Procs: create, get, set
Objects: Textsw

**TEXTSW_MEMORY_MAXIMUM**

Specifies how much memory to use when not editing files (e.g., editing in memory). This attribute only takes effect at create time or after the window is reset with textsw_reset. The lower bound is 1K bytes, which is silently enforced.

Argument: int
Default: 20000
Procs: create, get, set
Objects: Textsw
See Also: 8.4.8

**TEXTSW_MODIFIED**

Specifies whether the textsw has been modified.

Argument: Bool
Procs: get
Objects: Textsw
See Also: 8.4.1

**TEXTSW_MULTI_CLICK_SPACE**

Specifies the maximum number of pixels between successive mouse clicks to still have the clicks considered a multi-click.

Argument: int
Default: 4
Procs: create, get, set
Objects: Textsw
TEXTSW_MULTI_CLICK_TIMEOUT
Specifies the maximum number of milliseconds between successive mouse clicks to still have the
clicks considered a multi-click.
Argument: int
Default: 390
Procs: create, get, set
Objects: Textsw

TEXTSW_NOTIFY_PROC
Names a notify procedure.
Argument: void (*notify_proc) ()
Default: NULL
Procs: create, set
Objects: Textsw
Callback:

void
 notify_proc(textsw, avlist)
 Textsw textsw
 Attr_avlist avlist

Notify procedure installed by application using TEXTSW_NOTIFY_PROC.
See Also: 8.12

TEXTSW_READ_ONLY
If TRUE, prevents editing of the displayed text. If another file is loaded in, read-only status is turned
off again.
Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Textsw

TEXTSW_STATUS
If set, specifies the address of a variable of type Textsw_status. A value that reflects what hap-
pened during the call to xv_create is then written into it. Note that this attribute must appear in the
attribute list before the operation whose status you want to determine.
Argument: Textsw_status *
Default: None
Procs: create
Objects: Textsw
See Also: 8.2

TEXTSW_STORE_CHANGES_FILE
Controls whether the target filename given to textsw_store() to save the current contents to a
file changes the name of the file being edited (TEXTSW_FILE).
Argument: Bool
Default: TRUE
Procs: create, set
Objects: Textsw
See Also: 8.5.1
TEXTSW_SUBMENU_EDIT
Returns the submenu associated with the text pane menu Edit submenu.
Argument: Menu
Proc: get
Objects: Textsw

TEXTSW_SUBMENU_FILE
Returns the submenu associated with the text pane menu File submenu.
Argument: Menu
Proc: get
Objects: Textsw

TEXTSW_SUBMENU_FIND
Returns the submenu associated with the text pane menu Find submenu.
Argument: Menu
Proc: get
Objects: Textsw

TEXTSW_SUBMENU_VIEW
Returns the submenu associated with the text pane menu View submenu.
Argument: Menu
Proc: get
Objects: Textsw

TEXTSW_UPPER_CONTEXT
Specifies the minimum number of lines to maintain between the start of the selection and top of view.
A value of −1 means defeat the normal actions.
Argument: int
Default: 2
Proc: create, get, set
Objects: Textsw

TTY_ARGV
The command, specified as an argument vector, that the tty subwindow executes. Using the value
TTY_ARGV_DO_NOT_FORK lets a user start a tty subwindow without forking off a shell.
Argument: char**
Default: None
Proc: create, set
Objects: Textsw
See Also: 9.4, 9.1

TTY_CONSOLE
If TRUE, tty subwindow grabs console output.
Argument: Bool
Default: FALSE
Proc: set, create
Objects: Tty

XView Attributes 147
**TTY_PAGE_MODE**

If TRUE, output will stop after each page.

Argument: Bool
Default: FALSE
Procs: create, get, set
Objects: Tty

**TTY_PID**

The process ID of the program being run in the tty subwindow.

Argument: int
Default: None
Procs: create, get, set
Objects: Tty
See Also: 9.3

**TTY_QUIT_ON_CHILD_DEATH**

If TRUE, window_done is called on the parent frame of the tty window when its child terminates.

Argument: Bool
Default: FALSE
Procs: set, create
Objects: Tty
See Also: 9.3

**TTY_TTY_FD**

Gets the file descriptor of the pseudo-tty associated with the tty subwindow.

Argument: int
Procs: get
Objects: Tty
See Also: 9.4

**WIN_ALARM**

Rings the bell for that window.

Argument: No value
Default: off
Procs: set, create
Objects: Xv_Window
See Also: WIN_ALARM_DATA

**WIN_ALARM_DATA**

Gets the number of flashes and number of beeps used as alarm. (Currently always returns the same value.)

Argument: Xv_opaque
Default: 1 beep, 1 flash
Procs: get
Objects: Xv_Window
See Also: WIN_ALARM
WIN_BACKGROUND_COLOR
Specifies the background color of a window as an index into the colormap segment associated with the window.
Argument: int
Default: 0
Procs: create, get, set
Objects: Xv_Window
See Also: 4.7, 21.3.1.1, WIN_CMS, WIN_FOREGROUND_COLOR

WIN_BACKGROUND_PIXMAP
Specifies the background pixmap of a window.
Argument: Pixmap
Default: NULL
Procs: create, get, set
Objects: Xv_Window

WIN_BELOW
Causes the window to be positioned below the sibling window given as the value. Restricted to windows with the same immediate parent (i.e., subwindows). Does not affect the xv_x of the window.
Argument: Xv_Window
Default: N/A
Procs: set, create
Objects: Xv_Window

WIN_BIT_GRAVITY
Sets the Xlib-specific “bit gravity” on the underlying X window associated with the object. See <X11/X.h> for a list of legal values.
Argument: int
Default: NorthWestGravity
Procs: create, get
Objects: Xv_Window
See Also: 5.3, CANVAS_FIXED_IMAGE, CANVAS_RESIZE_PROC

WIN_BORDER
Controls whether a window has a border.
Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Xv_Window

WIN_CLIENT_DATA
Specifies an arbitrary value to be attached to a window.
Argument: caddr_t
Default: None
Procs: create, get, set
Objects: Xv_Window
See Also: 7.19.1, XV_KEY_DATA
**WIN_CMD_LINE**

Lets an application set the command-line options that can be used to (re)start it. The options passed, in addition to XView options are stored on a property called `WM_COMMAND` on the frame window. (The program `xprop` can be used to display a window’s properties). Only one base frame window of the application needs to have this property set. This property is read possibly by a session manager to restart clients. Setting this attribute to -1 prevents any command-line option information from being saved on the frame. If there are two or more base frames in the application, the second and subsequent base frames should set their `WIN_CMD_LINE` attributes to -1 if they want to avoid multiple invocations of the same application by the session manager.

The string passed is copied and cached on the frame.

**Warning:** Usage of this attribute will make the application non-ICCCM compliant. Use `FRAME_WM_COMMAND_ARGCARGV` instead. Intermixing the usage of `WIN_CMD_LINE` and `FRAME_WM_COMMAND_*` attributes has unpredictable results.

**Argument:** char *

**Default:** None

**Procs:** create, get, set

**Objects:** Xv_Window

**See Also:** `FRAME_WM_COMMAND_ARGCARGV`, `FRAME_WM_COMMAND_ARGC`, `FRAME_WM_COMMAND_ARGV`, `FRAME_WM_COMMAND_STRINGS`, Chapter 12, Interclient Communication, in Xlib Programming Manual.

**WIN_CMS**

Window-based objects use colormap segments to get their colors. These objects get a default colormap segment when they are created, but you can assign a new one using this attribute.

**Argument:** Cms

**Default:** Depends on `WIN_INHERIT_COLORS`, typically inherits parent’s cms

**Procs:** create, get, set

**Only use create and get for a FILE_CHOOSER.**

**Objects:** Xv_Window

**See Also:** 21.5, 21.2.2, 21.1, `WIN_FOREGROUND_COLOR`, `WIN_BACKGROUND_COLOR`

**WIN_CMS_DATA**

This attribute is obsolete; where possible, the Cms package should be used. It specifies the data for the colormap segment associated with the window. The data is written into the currently allocated colormap segment. If a new segment is desired, set it using `WIN_CMS_NAME` first.

**Argument:** Xv_cmsdata *

**Default:** None

**Procs:** create, get, set

**Objects:** Xv_Window

**WIN_CMS_NAME**

This attribute is obsolete; where possible, the Cms package should be used. Specifies the colormap segment to be associated with the window.

**Argument:** char *

**Default:** None

**Procs:** create, get, set

**Objects:** Xv_Window

**See Also:** 21.2.2
WIN_COLLAPSE_EXPOSURES
Collapses contiguous multiple Exposure (and GraphicsExpose) events destined for the same
window that are grouped by the count field in the X Expose event.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Xv_Window
See Also: 5.3

WIN_COLUMN_GAP
Specifies the gap between columns in the window.

Argument: int
Default: 0
Procs: create, get, set
Objects: Xv_Window

WIN_COLUMN_WIDTH
Specifies the width of a column in the window.

Argument: int
Default: Font's default width
Procs: create, get, set
Objects: Xv_Window

WIN_COLUMNS
Specifies the window’s width (including left and right margins) in columns relative to the width of the
window’s font. WIN_COLUMNS is not a valid xv_create() attribute for a Textsw. Issue a separate
xv_set() call after the xv_create() in order to set this attribute on a Textsw.

Argument: int
Default: Varies based on font, typically 80 columns
Procs: create, get, set
Objects: Xv_Window

WIN_CONSUME_EVENT
Specifies that the window will accept an event of type specified. The event is appended to the current
input mask.

Argument: Event code (defined in <xview/win_input.h>)
Default: Varies from package to package
Procs: create, get, set
Objects: Xv_Window
See Also: 6.3.2, 5.7.1, WIN_CONSUME_X_EVENT_MASK,
WIN_CONSUME_EVENTS, WIN_IGNORE_EVENT, WIN_IGNORE_EVENTS,
WIN_X_EVENT_MASK

WIN_CONSUME_EVENTS
Specifies a NULL-terminated list of event types that this window accepts. The event is appended to the
current input mask.

Argument: List of XView events
Default: Varies from package to package
Procs: create, set
Objects: Xv_Window
See Also: 6.3.2, WIN_CONSUME_X_EVENT_MASK, WIN_CONSUME_EVENT,
WIN_IGNORE_EVENT, WIN_IGNORE_EVENTS, WIN_X_EVENT_MASK
**WIN_CONSUME_X_EVENT_MASK**

The input mask is specified using X event masks found in `<X11/X.h>`. The event is appended to the current input mask.

Argument: unsigned long
Default: Varies from package to package
Procs: create, get, set
Objects: Xv_Window
See Also: 6.3.1, WIN_CONSUME_EVENT, WIN_CONSUME_X_EVENTS, WIN_IGNORE_EVENT, WIN_IGNORE_EVENTS, WIN_X_EVENT_MASK

**WIN_CURSOR**

Specifies the window’s cursor. You must supply the handle of an XView window as the parent parameter when getting WIN_CURSOR. Getting WIN_CURSOR on the root window returns NULL.

Argument: Xv_Cursor
Default: Default X server cursor
Procs: create, get, set
Objects: Xv_Window
Usage:

```c
default_cursor = (Xv_cursor)xv_get(paint_win, WIN_CURSOR);

xv_set(paint_win, WIN_CURSOR, default_cursor, NULL);
```

**WIN_DEPTH**

This attribute is obsolete. Use XV_DEPTH instead. Specifies the pixel depth of the window.

Argument: int
Default: Function of the screen
Procs: create, get
Objects: Xv_Window
See Also: 21.8, 2.4.2

**WIN_DYNAMIC_VISUAL**

This attribute is obsolete. Use the attribute XV_VISUAL to specify the visual used in the creation of the window or colormap segment.

Argument: Bool
Default: FALSE
Procs: create
Objects: Xv_Window

**WIN_EVENT_PROC**

Specifies a callback procedure where window events are delivered.

Argument: void (*event_proc) ()
Default: None
Procs: create, get, set
Objects: Xv_Window
Callback:

```c
void
    event_proc(window, event, arg)
    Xv_Window       window;
    Event           *event;
    Notify_arg      arg;
```

**WIN_FIT_HEIGHT**
Causes the window to shrink or expand its height according to the window’s contents leaving a margin specified by the value given. Typically used with panels to fit panel items.

**Argument:** int
**Default:** Depends on window size
**Procs:** create, get, set
**Objects:** Xv_Window
**See Also:** window_fit() macro in Section 3, Procedures and Macros.

**WIN_FIT_WIDTH**
Causes the window to shrink or expand its width according to the window’s contents leaving a margin specified by the value given.

**Argument:** int
**Default:** Depends on window size
**Procs:** create, get, set
**Objects:** Xv_Window
**See Also:** window_fit() macro in Section 3, Procedures and Macros.

**WIN_FOREGROUND_COLOR**
Specifies the foreground color of a window as an index into the colormap segment associated with the window.

**Argument:** int
**Default:** size ±1 (where size is the number of colors in the colormap segment associated with the window)
**Procs:** create, get, set
**Objects:** Xv_Window
**See Also:** 4.7, 21.3.1.1, WIN_CMS, WIN_BACKGROUND_COLOR

**WIN_FRAME**
Returns the window’s frame.

**Return Type:** Frame
**Default:** N/A
**Procs:** get
**Objects:** Xv_Window

**WIN_FRONT**
Indicates that the window should move to the front of the stacking order. Setting this attribute does not map the window. Use XV_SHOW to raise and map a window.

**Argument:** No value
**Procs:** create, set
**Objects:** Xv_Window

**WIN_GRAB_ALL_INPUT**
Specifies that the window will get all events regardless of location of the pointer. Performs a grab of the keyboard and the pointer.

**Argument:** Bool
**Default:** FALSE
**Procs:** set, create
**Objects:** Xv_Window
**See Also:** 6.6.1.2, FULLSCREEN attributes
WIN_HORIZONTAL_SCROLLBAR
This attribute is obsolete. Use OPENWIN_HORIZONTAL_SCROLLBAR instead. It is a handle to the horizontal scrollbar for that window.

Argument: Scrollbar
Default: NULL
Procs: create, get, set
Objects: Xv_Window

WIN_IGNORE_EVENT
Specifies that the window will not receive this event. (Certain events cannot be ignored.)

Argument: XView event code (see <xview/win_input.h>)
Default: None
Procs: create, set
Objects: Xv_Window
See Also: 6.3.2, WIN_CONSUME_EVENT, WIN_CONSUME_X_EVENT_MASK,
WIN_CONSUME_EVENTS, WIN_IGNORE_EVENTS

WIN_IGNORE_EVENTS
Specifies a NULL-terminated list of events that this window will not receive. (Certain events cannot be ignored.)

Argument: List of event codes (see <xview/win_input.h>)
Default: None
Procs: set, create
Objects: Xv_Window
See Also: 6.3.2, WIN_CONSUME_EVENT, WIN_CONSUME_X_EVENT_MASK,
WIN_CONSUME_EVENTS, WIN_IGNORE_EVENT, WIN_IGNORE_EVENTS

WIN_IGNORE_X_EVENT_MASK
Prevents the specified event masks from being delivered to the event handler. The input mask is specified using X event masks found in <XI1/X.h>.

Argument: unsigned long
Default: Varies from package to package
Procs: create, get, set
Objects: Xv_Window
See Also: 6.3.1, WIN_CONSUME_X_EVENT_MASK, WIN_EVENT_PROC,
WIN_IGNORE_EVENT, WIN_IGNORE_EVENTS, WIN_X_EVENT_MASK

WIN_INHERIT_COLORS
This attribute specifies whether the window should inherit the colors used by its parent. If TRUE, the window will inherit the WIN_CMS, XV_VISUAL, WIN_FOREGROUND_COLOR, and WIN_BACKGROUND_COLOR attributes of its parent. If FALSE, it will use the defaults for all of these attributes.

Argument: Bool
Default: TRUE
Procs: create, get, set
Objects: Xv_Window
**WIN_INPUT_MASK**

Specifies the window’s input mask. This overwrites the current mask.

- **Argument:** Inputmask *
- **Default:** Varies from package to package
- **Procs:** create, get, set
- **Objects:** Xv_Window
- **See Also:** 6.8

**WIN_INPUT_ONLY**

Specifies that a window is an input only window; although it has other window properties, it cannot be drawn into.

- **Argument:** No value
- **Default:** Input/Output Window
- **Procs:** create
- **Objects:** Xv_Window

**WIN_IS_CLIENT_PANE**

When used with `xv_create()`, the attribute has no value; using the attribute sets it to TRUE. When used, the window is considered to be an OPEN LOOK GUI application-specific pane which can override resources, such as fonts. Currently, the only packages that support this are text and tty/term.

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get
- **Objects:** Xv_Window

**WIN_KBD_FOCUS**

This attribute is for SunView compatibility. For more information, refer to the manual *Converting SunView Applications*.

**WIN_MAP**

Indicates whether to map or unmap the window. Setting this attribute to TRUE maps the window without changing the stacking order. This does not imply raised (to the top of the window tree). Use `XV_SHOW` to raise and map a window.

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get, set
- **Objects:** Xv_Window
- **See Also:** WIN_FRONT, XV_SHOW

**WIN_MENU**

This attribute is obsolete. The window manager now specifies the window’s menu.

**WIN_MESSAGE_DATA**

Used to access the data portion of a window’s client message.

- **Return Type:** char
- **Procs:** get
- **Objects:** Xv_Window
- **See Also:** 6.7.1, WIN_MESSAGE_FORMAT, WIN_MESSAGE_TYPE
WIN_MESSAGE_FORMAT
Used to access the format portion of a window’s client message.
Return Type: unsigned char
Procs: get
Objects: Xv_Window
See Also: 6.7.1, WIN_MESSAGE_DATA, WIN_MESSAGE_TYPE

WIN_MESSAGE_TYPE
Used to access the type portion of a window’s client message.
Return Type: Atom
Procs: get
Objects: Xv_Window
See Also: 6.7.1, WIN_MESSAGE_FORMAT, WIN_MESSAGE_DATA

WIN_MOUSE_XY
Warps the mouse pointer to the specified position. Returns a static Rect on xv_get.
Argument: int, int
Default: N/A
Procs: create, get, set
Objects: Xv_Window
See Also: 13.3

WIN_NO_CLIPPING
This attribute is for SunView compatibility. For more information, refer to the manual Converting SunView Applications.

WIN_PARENT
Specifies the window’s parent in the window tree. This attribute effectively reparents a window on set.
Argument: Xv_Window
Default: None
Procs: create, get, set
Objects: Xv_Window

WIN_PERCENT_HEIGHT
This attribute is for SunView compatibility. For more information, refer to the manual Converting SunView Applications.

WIN_PERCENT_WIDTH
This attribute is for SunView compatibility. For more information, refer to the manual Converting SunView Applications.

WIN_RECT
Specifies the bounding box of a window. Returns a static Rect on xv_get.
Argument: Rect *
Default: NULL
Procs: create, get, set
Objects: Xv_Window
**WIN RETAINED**

Hint to the server to maintain backing store for this window. The server may not honor this request.

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get, set
- **Objects:** Xv_Window
- **See Also:** 5.5

**WIN_RIGHT_OF**

Causes a sibling window to be laid out just to the right of the window given as the value. Restricted to windows that share the same immediate parent. Does not set XV_Y.

- **Argument:** Xv_Window
- **Default:** Arbitrary
- **Procs:** create, set
- **Objects:** Xv_Window
- **See Also:** WIN BELOW

**WIN_ROW_GAP**

Specifies the gap between rows in the window.

- **Argument:** int
- **Default:** 0
- **Procs:** create, get, set
- **Objects:** Xv_Window

**WIN_ROW_HEIGHT**

Specifies the height of a row in the window.

- **Argument:** int
- **Default:** Font's default height
- **Procs:** create, get, set
- **Objects:** Xv_Window

**WIN_ROWS**

Specifies the window’s height (including top and bottom margins) in rows. WIN_ROWS is not a valid xv_create() attribute for a Textsw. Issue a separate xv_set() call after the xv_create() in order to set this attribute on a Textsw.

- **Argument:** int
- **Default:** 34
- **Procs:** create, get, set
- **Objects:** Xv_Window

**WIN_SAVE_UNDER**

Provides a hint to the server about whether or not the screen area beneath a window should be saved while the window, such as a pop-up menu, is mapped. This is not the same as WIN_RETAINED.

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** set
- **Objects:** Xv_Window
WIN_SCREEN_RECT
Returns the bounding box of the screen containing the window. Data points to per-process static storage.

Argument: Rect *
Default: Screen dependent
Procs: get
Objects: Xv_Window

WIN_SET_FOCUS
Sets the input focus to this window, if possible. The X protocol restricts unmapped windows from holding the input focus. The window must select for KBD_USE and KBD_DONE events.

Argument: No value
Default: None
Procs: create, set
Objects: Xv_Window

WIN_SOFT_FNKEY_LABELS
Assigns labels for the soft function keys. The value for the WIN_SOFT_FNKEY_LABELS is a string of 12 labels separated by newline characters “\n”.

Argument: char *
Procs: get, set
Usage:

```c
canvas = (Canvas) xv_create (frame,CANVAS,
    CANVAS_X_PAINT_WINDOW,TRUE,
    NULL);

xv_set(canvas_paint_window(canvas),
    WIN_SOFT_FNKEY_LABELS,"Red\nGreen\nBlue\n..."
    WIN_EVENT_PROC, my_Event_proc,
    NULL);
```

In the above example, whenever the canvas gets the input focus, the soft function key labels will be updated to “Red Green Blue ...”. To display the soft function key panel, select the Function Keys item from the Workspace Utilities menu.

See Also: 6.12.1

WIN_TOP_LEVEL
Returns whether the window is the child of another window or is a frame.

Return Type: Bool
Default: N/A
Procs: get
Objects: Xv_Window
**WIN_TOP_LEVEL_NO_DECOR**
Controls whether this window is or is not controlled by the window manager. (This controls the OverrideRedirect flag.)

- **Argument:** Bool
- **Default:** FALSE
- **Procs:** create, get, set
- **Objects:** Xv_Window
- **See Also:** 4.10

**WIN_TRANSPARENT**
Specifies that the window’s background pixmap should be be transparent (set to none). For more information, on the background pixmap refer to Chapter 4, Frames, of the XView Programming Manual.

- **Argument:** No value
- **Default:** Not transparent
- **Procs:** create
- **Objects:** Xv_Window

**WIN_VERTICAL_SCROLLBAR**
This attribute is obsolete. Use OPENWIN_VERTICAL_SCROLLBAR instead. Specifies that scrollbar orientation is vertical.

- **Argument:** Scrollbar
- **Default:** None
- **Procs:** create, get, set
- **Objects:** Xv_Window

**WIN_WINDOW_GRAVITY**
Defines how the window should be repositioned if its parent is resized. See <X11/X.h> for legal values.

- **Argument:** int
- **Default:** NorthWestGravity
- **Procs:** create, get, set
- **Objects:** Xv_Window
- **See Also:** 5.3, WIN_BIT_GRAVITY

**WIN_X_COLOR_INDICES**
Translates the logical indices of the window’s colormap segment (from 0 to size-1) into the actual indices into the colormap used by the window.

- **Argument:** unsigned long *
- **Procs:** get
- **Objects:** Xv_Window
- **See Also:** 21.3.0.1

**WIN_X_EVENT_MASK**
Expect an X event mask. Acts in the same manner as WIN_CONSUME_X_EVENT_MASK; but instead of appending to the current input mask, replaces it with a new mask.

- **Argument:** unsigned long
- **Default:** Varies from package to package
- **Procs:** create, get, set
- **See Also:** WIN_CONSUME_EVENTS, WIN_CONSUME_X_EVENT_MASK
**XV_APP_NAME**
Sets the string to be used by XView as the application’s name. (Currently this is only for the help package). This also allows for more than one help file while still displaying the same name in the spot help header window.

Argument: char *
Default: argv[0]
Procs: set, get
Objects: Server
Usage:

```c
server = xv_init(XV_INIT_ARGS, argc, argv, 0);
xv_set(server,
      XV_APP_NAME, mailtool, NULL);

/* or (for localized app name) */
xv_set(server,
      XV_APP_NAME, gettext(mailtool), NULL);
```

**XV_AUTO_CREATE**
Specifies whether to create an object not found by xv_find().

Argument: Bool
Default: TRUE
Procs: find
Objects: Cms, Font, Menu_item
See Also: 25.9.2

**XV_BOTTOM_MARGIN**
Specifies the margin at the bottom of an object.

Argument: int
Default: Varies with object
Procs: create, get, set

**XV_DEPTH**
Specifies the pixel depth of the object.

Argument: int
Default: Function of the screen
Procs: create, get
Objects: Xv_Window

**XV_DISPLAY**
Returns the X Display data structure. The structure returned is maintained by the toolkit and should not be freed.

Argument: Display *
Default: Display structure for specified object
Procs: get
Objects: All Drawable objects and its subclasses
See Also: 15.1
XV_ERROR_PROC
The application’s XView error handler is called as a result of xv_error() being called.

Argument: int (*error_proc)()
Default: None
Procs: xv_init
Objects: N/A
Callback:

    int
    error_proc(object, avlist)
    Xv_object object;
    Attr_avlist avlist;

See Also: 24.2, xv_X_ERROR_PROC

XV_FOCUS_ELEMENT
Set focus on the first (0) or last (-1) element in the pane.

Argument: int
Valid Values: {0, -1}
Procs: set
Objects: Xv_Window

XV_FONT
Specifies an object’s font. Use only with xv_create() if you are setting a panel’s font.

Argument: Xv_Font
Default: lucida medium
Procs: create, get, set
    Only use create and get for a FILE_CHOOSER.
Object: Generally, all

XV_HEIGHT
Specifies the height of an object in pixels.

Argument: int
Default: N/A
    Defaults to a pre-determined number of rows and columns for a FILE_CHOOSER object.
    Should not be set at create time for a File Chooser.
Procs: create, get, set
Objects: All
See Also: 15.4

XV_HELP_DATA
Specifies the help string used by the help package to display on-line help. The text string has the form file:keyword. The help package looks for the key keyword in the file $HELPPATH/file.info. In the special case for a Textsw, set XV_HELP_DATA on the Textsw’s view which you can get with the following call:

    xv_get(textsw, OPENWIN_NTH_VIEW, NULL);

Argument: char *
Default: No help available notice
Procs: create, get, set
Objects: All
See Also: 23.1
**XV_INIT_ARGC_PTR_ARGV**

Interprets command-line args. Strips generic toolkit command-line arguments out of argv and decrements argc accordingly.

Argument 1: `int *
Argument 2: `char **
Default: None
Procs: xv_init
Objects: N/A
Usage:

```c
main(argc, argv
   int argc;
   char **argv;
   {
      server = xv_init (XV_INIT_ARGC_PTR_ARGV,
                     &argc, argv, NULL);
   }
```

See Also: 2.6.1, XV_INIT_ARGS

**XV_INIT_ARGS**

Interprets command-line arguments. Does not strip generic toolkit command-line arguments out of argv.

Argument 1: `int
Argument 2: `char **
Default: None
Procs: xv_init
Objects: N/A
Usage:

```c
main(argc, argv
   int argc;
   char **argv;
   {
      server = xv_init (XV_INIT_ARGS, argc, argv, NULL);
   }
```

See Also: 2.6.1, XV_INIT_ARGC_PTR_ARGV

**XV_INSTANCE_NAME**

The attribute XV_INSTANCE_NAME is used to associate an instance name with an XView object. The instance name is used to construct the resource name used by the resource manager to perform lookups. The resource name is constructed by concatenating the instance names of all objects in the current object’s lineage, starting with the name of the application or whatever was passed in with the -name command-line option, ending with the XView attribute name. The XView attribute name remains in lowercase. XV_INSTANCE_NAME is normally used with XV_USE_DB.

Argument: `char *
Default: NULL
Procs: create,set
Usage:

```c
Frame frame;
Panel panel;
frame = (Frame)xv_create(NULL, FRAME,
    XV_INSTANCE_NAME, "base_frame",
    NULL);
panel = (Panel)xv_create(frame, PANEL,
    XV_INSTANCE_NAME, "panel",
    XV_USE_DB,
    XV_WIDTH, 100,
    XV_HEIGHT, 200,
    NULL,
    NULL);
```

Above, assume the name of the application is `app`, the resource names constructed for lookup of the width and height of the panel are:

```c
app.base_frame.panel.xv_width
app.base_frame.panel.xv_height
```

Entries in the resource manager could look like:

```c
app.base_frame.panel.xv_width:400
app.base_frame.panel.xv_height:500
```

If these entries were not present in the resource manager, the width and height of the panel would take the default values of 100 and 200, respectively.

See Also: 22.3.2.3, 17.3, XV_USE_DB

**XV_KEY_DATA**

Stores a 32-bit data value on an object. You may set multiple `XV_KEY_DATA` values on objects by using different keys. The key specified should be a unique number (see `xv_unique_key()` in Section 3, Procedures and Macros).

Arguments:

- **Argument 1:** `int`
- **Argument 2:** `XVOpaque`
- **Default:** None
- **Procs:** create, get, set
- **Objects:** All

Usage:

```c
xv_create(panel, PANEL_BUTTON,
    PANEL_BUTTON_LABEL, "Push Me",
    XV_KEY_DATA, PANEL_ITEM_KEY, "text",
    PANEL_NOTIFY_PROC, my_notify_proc
    NULL);
```

See Also: 7.19.1

**XV_KEY_DATA_REMOVE**

Causes the data associated with the specified key to be removed. If an `XV_KEY_DATA_REMOVE_PROC` is defined, it will be called.

Argument:

- **int**
- **Default:** None
- **Procs:** set
- **Objects:** All

See Also: 7.19.1, XV_KEY_DATA, XV_KEY_DATA_REMOVE_PROC
**XV_KEY_DATA_REMOVE_PROC**

Names the function that is called whenever the object that has key data attached to it is destroyed. This function should free the data associated with the key.

- **Argument 1:** int
- **Argument 2:** void(*xv_key_data_remove_proc)()
- **Default:** None
- **Procs:** create, get, set
- **Objects:** All
- **Callback:**
  
  ```c
  void
  xv_key_data_remove_proc(object, key, data)
  Xv_object object;
  int key;
  caddr_t data;
  ```

- **See Also:** 7.19.1

**XV_LABEL**

Specifies a frame’s header label or an icon’s label or simply associates a name to an object. XView copies the strings on set.

- **Argument:** char *
- **Default:** NULL
- **For a FILE_CHOOSER, defaults to the string “Open”, “Save” or “Save As”, depending on the value of the FILE_CHOOSER_TYPE attribute. Note that the the proper format is Application: Popupname.
- **Procs:** create, get, set
- **Objects:** All
- **See Also:** 4.2.2

**XV_LC_BASIC_LOCALE**

 Specifies the basic locale category, which sets the country of the user interface. This XView attribute can only be used in xv_init(), but can be queried on any XView object which is a subclass of window or server class via xv_get(). The value that is set using these attributes must be a valid locale name in the system. Use in situations where you want to force a program to operate in a certain locale. For example, if a program only works in French, then the locale attribute can be set so that it cannot be switched to another language.

**Warning:** This attribute should only be used for localization operations.

- **Argument:** char *
- **Default:** "C"
- **Procs:** xv_init(), get
- **Usage:**
  
  ```c
  xv_init(...,
  XV_LC_BASIC_LOCALE, "fr",
  XV_LC_DISPLAY_LANG, "C",
  NULL);
  ```

- **See Also:** 22.1.4, XV_LOCALE_DIR, Chapter 22, Internationalization, in XView Programming Manual.
**XV_LC_DISPLAY_LANG**

Specifies the display language locale category, which sets the language in which labels, messages, menu items, and help text are displayed. This XView attribute can only be used in `xv_init()`, but can be queried on any XView object which is a subclass of window or server class via `xv_get()`. The value that is set using these attributes must be a valid locale name in the system. Use in situations where you want to force a program to operate in a certain locale. For example, if a program only works in French, then the locale attribute can be set so that it cannot be switched to another language.

**Warning:** This attribute should only be used for localization operations.

Argument: `char *`
Default: "C"
Procs: `xv_init()`, `get`
See Also: 22.1.4, `XV_LOCALE_DIR`, Chapter 22, *Internationalization*

**XV_LC_INPUT_LANG**

Specifies the input language locale category, which sets the language used for keyboard input. This XView attribute can only be used in `xv_init()`, but can be queried on any XView object which is a subclass of window or server class via `xv_get()`. The value that is set using these attributes must be a valid locale name in the system. Use in situations where you want to force a program to operate in a certain locale. For example, if a program only works in French, then the locale attribute can be set so that it cannot be switched to another language.

**Warning:** This attribute should only be used for localization operations.

Argument: `char *`
Default: "C"
Procs: `xv_init()`, `get`
See Also: 22.1.4, `XV_LOCALE_DIR`, Chapter 22, *Internationalization*

**XV_LC_NUMERIC**

Specifies the numeric locale category, which defines the language used to format numeric quantities. This XView attribute can only be used in `xv_init()`, but can be queried on any XView object which is a subclass of window or server class via `xv_get()`. The value that is set using these attributes must be a valid locale name in the system. Use in situations where you want to force a program to operate in a certain locale. For example, if a program only works in French, then the locale attribute can be set so that it cannot be switched to another language.

**Warning:** This attribute should only be used for localization operations.

Argument: `char *`
Default: "C"
Procs: `xv_init()`, `get`
See Also: 22.1.4, `XV_LOCALE_DIR`, Chapter 22, *Internationalization*

**XV_LC_TIME_FORMAT**

Specifies the time format locale category, which defines the language used to format time and date. This XView attribute can only be used in `xv_init()`, but can be queried on any XView object which is a subclass of window or server class via `xv_get()`. The value that is set using these attributes must be a valid locale name in the system. Use in situations where you want to force a program to operate in a certain locale. For example, if a program only works in French, then the locale attribute can be set so that it cannot be switched to another language.

**Warning:** This attribute should only be used for localization operations.

Argument: `char *`
Default: "C"
Procs: `xv_init()`, `get`
See Also: 22.1.4, `XV_LOCALE_DIR`, Chapter 22, *Internationalization*
XV_LEFT_MARGIN
Specifies the margin at the left of the object.

Argument: int
Default: Varies with object
Procs: create, get, set

XV_LOCALE_DIR
This attribute specifies the location of the application’s locale specific files. At present, these include
the app-defaults and LC_MESSAGES directories.

The directory structure referenced by XV_LOCALE_DIR is:

<XV_LOCALE_DIR>/<locale>/LC_MESSAGES
<XV_LOCALE_DIR>/<locale>/app-defaults

<locale> is expanded differently depending on which internationalization feature is using XV_
LOCALE_DIR.

Argument: char *
Default: $OPENWINHOME/lib/locale
Procs: xv_init()
See Also: 22.3.2.1, 22.2.1.6, 22.1.2.1

XV_MARGIN
Specifies the offset from the border of this object.

Argument: int
Default: Varies with object
Procs: create, get, set

XV_NAME
Specifies an optional name for an object. In some cases, this attribute is used internally by packages.
For instance, the SERVER package sets the display connection name using this attribute.

Argument: char *
Default: Varies with object
Procs: create, get, set
Objs: All
See Also: 15.3.1

XV_OWNER
Returns the object’s owner. The object returned varies among packages.

Return Type: XvOpaque
Procs: get
Objs: All

XV_RECT
Specifies the object’s bounding box; that is, the smallest rectangle that contains the object. The
Rect* returned points to a per-process static storage and thus should not be freed.

Argument: Rect *
Default: None. Should not be set at create time for a File Chooser.
Procs: create, get, set
Objs: Window and its subclasses
See Also: 25.2.1
XV_RIGHT_MARGIN
Specifies the margin at the right of the object.
Argument: int
Default: Varies with object
Procs: create,get,set

XV_ROOT
Returns the root window for an object. (Object must be a window based.)
Argument: Xv_object
Default: Root window for default screen
Procs: get
Objects: Window and its subclasses

XV_SCREEN
Returns the screen object associated with this object.
Argument: Xv_screen
Default: Values for specified screen
Procs: get
Objects: Window and its subclasses
See Also: 15.2

XV_SHOW
Causes the object to be displayed or undisplayed. If object is a window, brings object to the top.
Argument: Bool
Default: Varies
For a FILE_CHOOSER object, it is not recommended to set XV_SHOW to TRUE at create time.
Procs: create,get,set
Objects: Generic
See Also: 4.3.1, WIN_MAP, WIN_FRONT

XV_TOP_MARGIN
Specifies the margin at the top of the object.
Argument: int
Default: Varies with object
Procs: create,get,set

XV_TYPE
Returns the package that belongs to an object.
Argument: Xv_pkg *
Default: N/A
Procs: get
Objects: All
See Also: 24.4.2
**XV_USAGE_PROC**

An application-defined routine that is called when an application is started with the `-help` option. Typically used to print a list of valid command-line options. This routine should not return. Applications that add additional command-line options to those already provided by XView should register an `XV_USAGE_PROC`. From within the `XV_USAGE_PROC`, the application should first print the application defined options and then call `xv_usage()` to allow XView to print the default command-line options.

**Argument:** `void (*xv_usage_proc)()`

**Default:** `xv_usage()` (XView defined usage procedure)

**Usage:**

```c
application_usage_proc(name)
char *name
{
    printf("...");
    xv_usage(name);
    exit(1);
}
main()
{
    xv_init(... XV_USAGE_PROC,
            application_usage_proc, ...);
}
```

**Callback:**

```c
void
usage_proc(application_name)
char *application_name
```

**See Also:** 2.6.1

**XV_USE_DB**

The attribute `XV_USE_DB` can be used to specify a set of attributes that are to be searched in the X11 resource manager database. `XV_USE_DB` takes a NULL-terminated list of attribute-value pairs as its values. During program execution, each attribute in this NULL-terminated list of attributes is looked up in the X11 resource manager database. If the attribute is not found in the database, then the value specified in the attribute-value pair is used as the default value.

The list of customizable attributes is shown below:

- `CANVAS_HEIGHT`
- `CANVAS_MIN_PAINT_HEIGHT`
- `CANVAS_WIDTH`
- `CANVAS_MIN_PAINT_WIDTH`
- `PANEL_CHOICE_NCOLS`
- `PANEL_CHOICE_NROWS`
- `PANEL_DROP_HEIGHT`
- `PANEL_DROP_WIDTH`
- `PANEL_EXTRA_PAINT_HEIGHT`
- `PANEL_EXTRA_PAINT_WIDTH`
- `PANEL_GAUGE_WIDTH`
- `PANEL_ITEM_X`
- `PANEL_ITEM_X_GAP`
- `PANEL_ITEM_Y`
- `PANEL_ITEM_Y_GAP`
- `PANEL_LABEL_WIDTH`
- `PANEL_LABEL_X`
- `PANEL_LABEL_Y`
- `PANEL_LIST_DISPLAY_ROWS`
- `PANEL_LIST_HEIGHT`
- `PANEL_LIST_WIDTH`
- `PANEL_MAX_VALUE`
- `PANEL_MIN_VALUE`
- `PANEL_NEXT_COL`
- `PANEL_NEXT_ROW`
- `PANEL_SLIDER_WIDTH`
- `PANEL_TICKS`
- `PANEL_VALUE_DISPLAY_LENGTH`
- `PANEL_VALUE_DISPLAY_WIDTH`
- `PANEL_VALUE_STORED_LENGTH`
- `PANEL_VALUE_X`
Frame frame;
frame = (Frame)xv_create(NULL, FRAME,
    XV_INSTANCE_NAME, "base_frame",
    XV_USE_DB,
    XV_WIDTH, 100,
    XV_HEIGHT, 200,
    NULL,
    NULL);

In this example, the value of the attributes XV_WIDTH and XV_HEIGHT for the frame object is searched in the X11 resource manager database. If not found, XV_WIDTH and XV_HEIGHT will take the default values of 100 and 200 respectively. See XV_INSTANCE_NAME for information regarding the actual resource name used for the search.

For more detailed information, refer to the Xlib Programming Manual.

See Also: 22.3.2.2, 17.3, XV_INSTANCE_NAME, Chapter 22, Internationalization

XV_USE_LOCALE
This attribute enables the internationalization features in XView. When this attribute is FALSE (default), internationalization features in XView are turned off. All internationalized applications must set this attribute to be TRUE. This attribute is only valid in the xv_init() function call. The format is as follows:

Frame frame;
frame = (Frame)xv_create(NULL, FRAME,
    XV_INSTANCE_NAME, "base_frame",
    XV_USE_DB,
    XV_WIDTH, 100,
    XV_HEIGHT, 200,
    NULL,
    NULL);

xv_init(XV_USE_LOCALE, TRUE,...);

See Also: 22.1.2

XV_VISUAL
This attribute specifies the exact visual that will be used in the creation of the window or colormap segment. The value is a pointer to a visual structure that one would normally get from XMatchVisualInfo or XGetVisualInfo. This attribute applies only to WINDOW and cms objects.

See Also: 21.8
**XV_VISUAL_CLASS**

This attribute specifies the class of the visual that will be used in the creation of the window or colormap segment. The value should be one of the following:

- StaticGray
- GrayScale
- StaticColor
- PseudoColor
- TrueColor
- DirectColor

If the server that the window is being created with does not support the visual class that is specified by this attribute, the library will use the default visual instead. This attribute applies only to WINDOW and CMS objects.

**Argument:** int  
**Default:** Varies with server (usually PseudoColor)  
**Procs:** create, get  
**See Also:** 21.8

**XV_WIDTH**

Specifies the width of the object in pixels.

**Argument:** int  
**Default:** None  
**Procs:** create, get, set  
**Objects:** Generic  
**See Also:** 15.4

**XV_X**

Specifies the x position of the object relative to its parent.

**Argument:** int  
**Default:** Varies with object  
**Procs:** create, get, set  
**See Also:** 7.4.2

**XV_X_ERROR_PROC**

Specifies the application’s Xlib error handler (different from XV_ERROR_PROC). This function should return XV_OK if the error is to be ignored by XView. Return XV_ERROR if XView is to process the X error and exit.

**Argument:** int (*x_error_proc) ()  
**Default:** None  
**Procs:** xv_init  
**Callback:**

```c
int
x_error_proc(display, event)
    Display *display;
    XErrorEvent *event;
```

**See Also:** 24.3, XV_ERROR_PROC
**XV_XID**
Returns the XID of the specified object such as the pixmap associated with a `Server_image` or the X Window associated with a canvas paint window.

Argument: int
Default: N/A
Procs: get
Objects: Drawable and its subclasses
See Also: 16.1

**XV_Y**
Specifies the y position of the object relative to its parent.

Argument: int
Default: Varies with object
Procs: create, get, set
Objects: Generic
See Also: 7.4.2
This page intentionally left blank

to preserve original page counts.
This page intentionally left blank

to preserve original page counts.
3
Procedures and Macros

This section lists the XView procedures and macros in alphabetical order. For each proce-
dure or macro, an explanation of its use is provided. Following each procedure is a synopsis
that includes the procedure’s parameters, if there are any. The parameters are described after
the synopsis.

**attr_create_list()**
Takes an attribute-value list and converts it into an array of `Attr_attributes`, also
known as an `Attr_avlist`. `ATTRIBUTE_STANDARD_SIZE`, defined in `<xview/attr.h>`,
defines how big an attribute-value list can get. This is the limit on how big an attri-
bute-value list can be created using the `ATTR_LIST` or `attr_create_list()` macros.

```c
Attr_avlist
attr_create_list(attrs)
```

The return value is used as the value for the attribute `ATTR_LIST`.

**CANVAS_EACH_PAINT_WINDOW()**
Macro providing built-in support for iteration across all the paint windows contained in
a given canvas. Allows you to perform operations on multiple paint windows for
which there are no canvas attributes.

**CANVAS_END_EACH**
Closes the loop started by `CANVAS_EACH_PAINT_WINDOW()`. These two macros are
meant to be used together.

**cursor_copy()**
Creates and returns a copy of `src_cursor`.

```c
Xv_Cursor
cursor_copy(src_cursor)
Xv_Cursor src_cursor;
```
**defaults_exists()**

This function returns TRUE if the resource exists in the database via `XrmGetResource()`.

```c
Bool
defaults_exists(name, class)
    char *name;
    char *class;
```

**defaults_exists()**

defaults_set_boolean() sets the resource to the value specified.

```c
void
defaults_set_boolean(resource, value)
    char *resource;
    Bool value;
```

**defaults_get_boolean()**

defaults_get_boolean() looks up the name-class pair in the resource database and returns TRUE if the value is one of:

- True
- Yes
- On
- Enabled
- Set
- Activated
- 1

It returns FALSE if the value is one of:

- False
- No
- Off
- Disabled
- Reset
- Cleared
- Deactivated
- 0

If the value is none of the previous, a warning message will be displayed and the default value will be returned. If the resource is not found, no error message is printed but the default value is still returned.

```c
Bool
defaults_get_boolean(name, class, default_value)
    char *name, *class;
    int default_value;
```
**defaults_get_character()**

`defaults_get_character()` looks up the name-class pair in the resource database and returns the resulting character value. If the resource is not found, then the default character value is returned.

```
char
defaults_get_character(name, class, default_char)
    char *name;
    char *class;
    char default_char;
```

**defaults_get_enum()**

`defaults_get_enum()` looks up the value associated with name and class and scans the pairs table and returns the associated value. If no match is found, an error is generated and the value associated with the last entry is returned. `defaults_get_enum()` calls `defaults_get_string()` and determines the value returned by calling `defaults_lookup()`, passing the returned string as the name parameter.

```
int
defaults_get_enum(name, class, pairs)
    char *name;
    char *class;
    Defaults_pairs *pairs;
```

**defaults_get_integer()**

`defaults_get_integer()` looks up the name-class pair in the resource database and returns the resulting integer value. If the database does not contain the resource, the default value is returned.

```
int
defaults_get_integer(name, class, default_value)
    char *name;
    char *class;
    int default_value;
```

**defaults_get_integer_check()**

`defaults_get_integer_check()` looks up the name-class pair in the resource database and returns the resulting integer value. If the value in the database is not between the values `minimum` and `maximum` (inclusive), an error message is printed and the default value is returned. If the resource is not found, no error message is printed but the default value is returned.

```
int
defaults_get_integer_check(name, class, default_value, minimum, maximum)
    char *name;
    char *class;
    int default_value;
    int minimum;
    int maximum;
```
defaults_get_string()  
defaults_get_string() returns the string value associated with the specified name-class pair in the resource database. If the resource is not found, the default string value is returned. defaults_get_string() returns a pointer into a static buffer maintained by the defaults package. Applications should not attempt to free this pointer. This buffer will be overwritten by the next call to the defaults package, so the application should maintain a copy if necessary.

    char *
defaults_get_string(name, class, default_str)
    char *name;
    char *class;
    char *default_str;

defaults_init_db()  
This function is called automatically by xv_init(), so it need not be called by your application. defaults_init_db() calls XrmInitialize().

    void
defaults_init_db()

defaults_load_db()  
defaults_load_db() loads the database residing in the specified filename or the server database if filename is NULL. The database found in filename is loaded via XrmGetFileDatabase() and is merged into the existing resource database via XrmMergeDatabases().

    void
defaults_load_db(filename)
    char *filename;

defaults_lookup()  
defaults_lookup() linearly scans the pairs array looking for name. The value associated with name is returned. The pairs array must contain a last element with a NULL name and a legal value associated with it. This value is returned if name does not match the name field of any of the elements in the pairs parameter.

    int
defaults_lookup(name, pairs)
    char *name;
    Defaults_pairs *pairs;

defaults_set_character()  
defaults_set_character() sets the resource to the character value.

    void
defaults_set_character(resource, character)
    char *resource;
    char character;
defaults_set_integer()
defaults_set_integer() sets the resource to the value specified.

void
defaults_set_integer(resource, value)
    char *resource;
    int value;

defaults_set_string()
defaults_set_string() sets the resource to the specified string.

void
defaults_set_string(resource, string)
    char *resource;
    char *string;

defaults_store_db()

void
defaults_store_db(filename)
    char *filename;

This function writes the database to the specified file via XrmPutFileDatabase(). This must be done in order to ensure that the database is accessible the next time the server is started.

dnd_decode_drop()

This function initiates a drag and drop data transfer using the selection mechanism. The drop event is decoded in three steps: First, the selection defined within the drop event is associated with the selection object passed into dnd_decode_drop(); second, the function sends an acknowledgement to the source, informing it that the transaction has been initiated; finally, if the drop-site is valid, the drop-site item that was dropped on is returned. Otherwise, if dnd_decode_drop() fails to initiate the drag and drop transaction, XV_ERROR is returned.

Xv_drop_site
dnd_decode_drop(sel_object, drop_event)
    Selection_requestor *sel_object;
    Event *drop_event;

sel_object is an instantiated selection requestor object. drop_event is a pointer to the event structure that contains the trigger event. This event is either the semantic event ACTION_DRAG_MOVE or ACTION_DRAG_COPY, indicating the type of the drag and drop: a move or a copy.
**dnd_done()**

This function is called by the application that is receiving the drop. It is called *after* the drop operation is completed. The function informs the toolkit that the drag and drop operation has been completed.

```c
    dnd_done(sel_object)
    Selection_requestor sel_object;
```

`sel_object` is the `Selection_requestor` object that was previously passed into `dnd_decode_drop()`.

**dnd_is_forwarded()**

Allows applications to detect when a drop event has been forwarded from some other drop-site. Generally this happens when the user drops on the window manager’s decor window or on an icon with the attribute `DROP_SITE_DEFAULT` set. The corresponding preview/drop event will have the flags field set with the `DND_FORWARDED` flag. This is tested for by using `dnd_is_forwarded(event)`.

In general, if the application handles previewing, it should check to see if the preview event was forwarded and not invert/highlight the drop site.

```c
    dnd_is_forwarded(event)
```

**dnd_is_local()**

Returns whether the `ACTION_DRAG_COPY` or `ACTION_DROP_MOVE` event was generated locally within the client.

```c
    dnd_is_local(event)
```

**dnd_send_drop()**

This initiates a drag. After the drag and drop object is created, an application calls `dnd_send_drop()`. This changes the cursor to the drag cursor, sends preview events to valid drop-sites and sends the trigger event (if a valid drop-site is dropped on). Trigger events are either `ACTION_DRAG_COPY` or `ACTION_DRAG_MOVE`.

A drag operation can be aborted by hitting the STOP key, or its equivalent. `dnd_send_drop()` reverts the drag cursor back to its normal state and returns `DND_ABORTED` when it detects the STOP key event. Application code needs to handle the `DND_ABORTED` after `dnd_send_drop()` returns. See `<xview/dragdrop.h>` for other valid return values.

```c
    int dnd_send_drop(object)
    Dnd object;

    object is a Dnd object that describes the source data (an instance of `DRAGDROP`).
```

**event_action()**

Returns the semantic action from the event structure. For example, when the user selects the SELECT button, it returns `ACTION_SELECT`. If there is no action associated with an event, `event_action()` returns the value returned by `event_id()` for the event. The macro is passed a pointer to an `Event` structure.
event_alt_is_down()
Returns TRUE or FALSE, depending on the state of the Alt key. The macro is passed a pointer to an Event structure.

event_button_is_down()
Returns TRUE or FALSE, depending on the state of mouse buttons. Used in conjunction with: event_is_button(), event_left_is_down(), event_middle_is_down(), event_right_is_down(). The macro is passed a pointer to an Event structure.

event_ctrl_is_down()
Returns TRUE or FALSE, depending on the state of the Control key. Indicates the state of the Control key. The macro is passed a pointer to an Event structure.

event_id()
Returns the actual event ID, such as MS_LEFT to indicate the left mouse button. The macro is passed a pointer to an Event structure. Use of the macro event_action(), which returns the semantic action, is recommended instead of event_id().

event_is_ascii()
Returns TRUE or FALSE, depending whether or not the event is an ASCII key. The macro is passed a pointer to an Event structure.

event_is_button()
Returns TRUE or FALSE, depending whether or not the event is a button event. This procedure is used with: event_button_is_down(), event_left_is_down(), event_middle_is_down(), and event_right_is_down(). The macro is passed a pointer to an Event structure.

event_is_down()
Returns TRUE or FALSE, depending on the state of a mouse button or keyboard key. The macro is passed a pointer to an Event structure. Also see the macro event_is_up().

event_is_iso()
Returns TRUE or FALSE, depending whether a key event is within the ISO character set. The macro is passed a pointer to an Event structure.

event_is_key_bottom()
Returns TRUE or FALSE, depending whether a key event is within the left function keys. The macro is passed a pointer to an Event structure.

event_is_key_left()
Returns TRUE or FALSE, depending whether a key event is within the left function keys. This is for function keys on keyboards that are sectioned into four sets of fifteen function keys. The macro is passed a pointer to an Event structure.
event_is_key_right()
  Returns TRUE or FALSE, depending whether a key event is within the right function keys. This is for function keys on keyboards that are sectioned into four sets of fifteen function keys. The macro is passed a pointer to an Event structure.

event_is_key_top()
  Returns TRUE or FALSE, depending whether a key event is within the top function keys. This is for function keys on keyboards that are sectioned into four sets of fifteen function keys. The macro is passed a pointer to an Event structure.

event_is_meta()
  Returns TRUE or FALSE, and determines whether event is a Meta key. The macro is passed a pointer to an Event structure.

event_is_string()
  Returns TRUE or FALSE, determines if a string is associated with the event. The macro is passed a pointer to an Event structure.

event_is_up()
  Returns TRUE or FALSE, depending on the state of the mouse button or keyboard in a particular event. The macro is passed a pointer to an Event structure. Also see the macro event_is_down().

event_left_is_down()
  Returns TRUE or FALSE, and determines the state of the left mouse button. Used to determine the state of particular buttons for a three button mouse. This procedure is used in conjunction with: event_button_is_down(), event_is_button(), event_middle_is_down(), and event_right_is_down(). The macro is passed a pointer to an Event structure.

event_meta_is_down()
  Indicates the state of the Meta key. The macro is passed a pointer to an Event structure.

event_middle_is_down()
  Returns TRUE or FALSE, and determines the state of the middle mouse button. This procedure is used with: event_button_is_down(), event_is_button(), event_left_is_down(), and event_right_is_down(). The macro is passed a pointer to an Event structure.

event_right_is_down()
  Returns TRUE or FALSE, and determines the state of the right mouse button. This procedure is used with: event_button_is_down(), event_is_button(), event_left_is_down(), and event_middle_is_down(). The macro is passed a pointer to an Event structure.

event_shift_is_down()
  Indicates the state of the Shift key. The macro is passed a pointer to an Event structure.
event_string()
Used to determine the string value associated with the event ID. For normal ASCII event codes, event_string() returns a string value of the key. However, keys, ASCII or not, may be rebound to a string using XRebindKeysym(). The macro is passed a pointer to an Event structure.

event_time()
Returns the time field from the event structure. Used to determine the time value associated with the event. The macro is passed a pointer to an Event structure.

event_window()
Gets the window in which a particular event took place. The macro is passed a pointer to an Event structure.

event_xevent()
Returns a pointer to the X event structure associated with an XView event structure. The macro is passed a pointer to an Event structure.

event_xevent_type()
Returns the type field of the X event structure. The macro is passed a pointer to an Event structure.

frame_get_rect()
Gets the rect of the frame. x, y is the upper-left corner of the window coordinate space. Width and height attempt to include the window manager decoration.

```c
void frame_get_rect(frame, rect)
    Frame frame;
    Rect *rect;
```

frame_set_rect()
Sets the rect of the frame. x, y is the upper-left corner of the window coordinate space. Width and height attempt to include the window manager decoration.

```c
void frame_set_rect(frame, rect)
    Frame frame;
    Rect *rect;
```

MENUITEM_SPACE
Used to create a blank menu item.

```c
menu_item = xv_create(NULL, MENUITEM_SPACE, NULL);
```
**menu_return_item()**

Predefined notify procedure which, if given as the value for MENU_NOTIFY_PROC, causes the menu_done_proc(), if any, to return the handle of the selected item.

```c
Menu_item
menu_return_item(menu, menu_item)
```

**menu_return_value()**

Predefined notify procedure which, if given as the value for MENU_NOTIFY_PROC, causes the menu_done_proc(), if any, to return the value of the selected item.

```c
Xv_opaque
menu_return_value(menu, menu_item)
```

**menu_show()**

Displays the specified menu.

```c
void
menu_show(menu, window, event, NULL)
```

**notice_prompt()**

This is an XView Version 2 compatibility procedure. If you are creating a new notice for XView Version 3, use the NOTICE package. Refer to Chapter 12, Notices, in the XView Programming Manual for details. To obtain more information on notice_prompt(), see Appendix B, The notice_prompt Function, in the XView Programming Manual.

The function displays the notice and does not return until the user pushes a button or until its trigger or the default action has been seen. It returns a value of NOTICE_FAILED if notice_prompt() fails for any reason; otherwise, it is equivalent to the ordinal value of the button which caused the return (i.e., button actually selected or default button if default action triggered return). The client_window should be the window for which the notice has been generated. This is important for fonts and positioning information. For explicit positioning of the notice, clients should specify NOTICE_FOCUS_XY. The event, if not NULL, will be completely filled in at the time the notice_prompt() returns.

The possible status values that may be returned from this function are:

- The int value passed with every NOTICE_BUTTON attribute.
- NOTICE_YES if a button created with NOTICE_BUTTON_YES is pushed.
- NOTICE_NO if a button created with NOTICE_BUTTON_NO is pushed.
• NOTICE_TRIGGERED if a trigger was used.
• NOTICE_FAILED if the notice failed to pop up.

```
int
notice_prompt(window, event, attributes)
    XV_Window window;
    Event *event;
    attribute-list attributes;
```

**notify_default_wait3()**
Predefined function you can register with the Notifier via the notify_set_wait3_func() call. Causes the required housekeeping to be performed on the process identified by pid when it dies. See the wait(2) man page for details of the wait and rusage structures.

```
Notify_value
notifiy_default_wait3(client, pid, status, rusage)
    Notify_client client;
    int pid;
    int *status;
    struct rusage *rusage;
```

**notify_dispatch()**
Provided to allow programs which are not notification-based to run in an event-driven environment. Called regularly from within the application’s main loop to allow the Notifier to go once around its internal loop and dispatch any pending events.

```
Notify_error
notify_dispatch()
```

**notify_do_dispatch()**
Enables implicit dispatching, in which the Notifier dispatches events from within calls to read(2) or select(2).

```
Notify_error
notify_do_dispatch()
```

**notify_enable_rpc_svc()**
If you use an RPC server that also needs to work with XView, use the XView function notify_enable_rpc_svc(), and do not call svc_run(). This function takes an int that tells the notifier whether it should handle RPC requests.

Using this approach, xv_main_loop() handles incoming RPC requests; dispatching them just as if svc_run() had been called.

If notify_enable_rpc_svc is enabled, the Notifier checks svc_fdset and calls svc_getreqset() if a descriptor is readable (a request is coming in). Performance is affected by the additional call to svc_getreqset().

```
void
notify_enable_rpc_svc(bool)
    int bool;
```
notify_flush_pending()
   Notifier removes client and flushes requests for it.

    void
    notify_flush_pending(nclient)
        Notify_client  nclient;

    See Also: notify_no_dispatch().

notify_get_destroy_func()
    Notify_func
    notify_get_destroy_func(client)
        Notify_client  client;

notify_get_event_func()
    Notify_func
    notify_get_event_func(client, when)
        Notify_client  client;
        int          when;

notify_get_exception_func()
    Notify_func
    notify_get_exception_func(client, fd)
        Notify_client  client;
        int          fd;

notify_get_input_func()
    Notify_func
    notify_get_input_func(client, fd)
        Notify_client  client;
        int          fd;

notify_get_itimer_func()
    Notify_func
    notify_get_itimer_func(client, which)
        Notify_client  client;
        int          which;

notify_get_output_func()
    Notify_func
    notify_get_output_func(client, fd)
        Notify_client  client;
        int          fd;
**notify_get_signal_func()**

```c
Notify_func
notify_get_signal_func(client, signal, mode)
    Notify_client client;
    int signal;
    Notify_signal_mode mode;
```

**notify_get_wait3_func()**

```c
Notify_func
notify_get_wait3_func(client, pid)
    Notify_client client;
    int pid;
```

**notify_interpose_destroy_func()**

Interposes `destroy_func()` in front of client’s destroy event handler.

```c
Notify_error
notify_interpose_destroy_func(client, destroy_func)
    Notify_client client;
    Notify_func destroy_func;
```

client is the handle of the Notifier client in front of which you are interposing.
destroy_func is a notify function to be called before the client’s destroy function.
The format for the destroy function is:

```c
Notify_value
destroy_func(client, status)
    Notify_client client;
    Destroy_status status;
```

**notify_interpose_event_func()**

Interposes `event_func()` in front of client’s event handler.

```c
Notify_error
notify_interpose_event_func(client, event_func, type)
    Notify_client client;
    Notify_func event_func;
    Notify_event_type type;
```

client is the handle of the Notifier client in front of which you are interposing.
event_func is the notify function to be called before the client’s event function.
type is either NOTIFY_SAFE or NOTIFY_IMMEDIATE. The format for the event function is:

```c
Notify_value
event_func(client, event, arg, type)
    Notify_client client;
    Notify_event event;
    Notify_arg arg;
    Notify_event_type type;
```
notify_interpose_exception_func()
Interposes exception_func in front of the client’s exception handler.

Notify_error
notify_interpose_exception_func(client, exception_func, fd)
    Notify_client client;
    Notify_func exception_func;
    int fd;

client is the handle of the Notifier client in front of which you are interposing. exception_func is the notify function to be called before the client’s exception function. type is either NOTIFY_SAFE or NOTIFY_IMMEDIATE. The format for the exception function is:

    Notify_value
    exception_func(client, fd)
        Notify_client client;
        int fd;

notify_interpose_input_func()
Interposes input_func in front of client’s Notifier.

Notify_error
notify_interpose_input_func(client, input_func, fd)
    Notify_client client;
    Notify_func input_func;
    int fd;

client is the handle of the Notifier client in front of which you are interposing. input_func is the notify function to be called before the client’s input function. The format for the input function is:

    Notify_value
    input_func(client, fd)
        Notify_client client;
        int fd;

notify_interpose_itimer_func()
Interposes the itimer_func in front of the client’s timeout event handler.

Notify_error
notify_interpose_itimer_func(client, itimer_func, which)
    Notify_client client;
    Notify_func itimer_func;
    int which;

client is the handle of the Notifier client in front of which you are interposing. itimer_func is the notify function to be called before the client’s timeout handler. which is either ITIMER_REAL or ITIMER_VIRTUAL. The format for the itimer function is:

    Notify_value
    itimer_func(client, which)
        Notify_client client;
        int which;
**notify_interpose_output_func()**

Interposes output_func in front of client’s Notifier.

```
Notify_error
notify_interpose_output_func(client, output_func, fd)
    Notify_client  client;
    Notify_func    output_func;
    int            fd;
```

*client* is the handle of the Notifier client in front of which you are interposing. *output_func* is the notify function to be called before the client’s output function.

The format for the output function is:

```
Notify_value
output_func(client, fd)
    Notify_client  client;
    int            fd;
```

**notify_interpose_signal_func()**

Interposes the signal_func() in front of the signal event handler.

```
Notify_error
notify_interpose_signal_func(client, signal_func, signal, mode)
    Notify_client  client;
    Notify_func    signal_func;
    int            signal;
    int            mode;
```

*client* is the handle of the Notifier client in front of which you are interposing. *signal_func* is the notify function to be called before the client’s signal handler. *signal* is the UNIX software interrupt. *mode* is either NOTIFY_ASYNC or NOTIFY_SYNC. The format for the signal function is:

```
Notify_value
signal_func(client, signal, mode)
    Notify_client  client;
    int            signal;
    int            mode;
```

**notify_interpose_wait3_func()**

Interposes the wait3_func in front of the Notifier.

```
Notify_error
notify_interpose_wait3_func(client, wait3_func, pid)
    Notify_client  client;
    Notify_func    wait3_func;
    int            pid;
```

*client* is the handle of the Notifier client in front of which you are interposing. *wait3_func* is the notify function to be called before the client’s wait3 function. The format for the wait3 function on BSD-based systems is:

```
Notify_value
wait3_func(client, pid, status, rusage)
    Notify_client  client;
    int            pid;
```
union wait *status;
struct rusage *rusage;

The format for the wait3 function on SYSV-based systems is:

```
Notify_value
wait3_func(client, pid, status, rusage)
    Notify_client client;
    int pid;
    int *status;
    struct rusage *rusage;
```

**notify_itimer_value()**

Returns the current state of an interval timer for `client` in the structure pointed to by `value`. The `which` parameter is either `ITIMER_REAL` or `ITIMER_VIRTUAL`.

```
Notify_error
notify_itimer_value(client, which, value)
    Notify_client client;
    int which;
    struct itimerval *value;
```

**notify_next_destroy_func()**

Calls the next destroy event handler for `client`.

```
Notify_value
notify_next_destroy_func(client, status)
    Notify_client client;
    Destroy_status status;
```

`status` is one of:

- DESTROY_PROCESS_DEATH
- DESTROY_CHECKING
- DESTROY_CLEANUP
- DESTROY_SAVE_YOURSELF

**notify_next_event_func()**

Calls the next event handler for `client`.

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

`type` is either `NOTIFY_SAFE` or `NOTIFY_IMMEDIATE`.

**notify_next_exception_func()**

```
Notify_value
notify_next_exception_func(client, fd)
    Notify_client client;
    int fd;
```
**notify_next_input_func()**

Notify_value

notify_next_input_func(client, fd)
  Notify_client client;
  int fd;

**notify_next_itimer_func()**

Notify_value

notify_next_itimer_func(client, which)
  Notify_client client;
  int which;

**notify_next_output_func()**

Notify_value

notify_next_output_func(client, fd)
  Notify_client client;
  int fd;

**notify_next_signal_func()**

Notify_value

notify_next_signal_func(client, signal, mode)
  Notify_client     client;
  int              signal;
  Notify_signal_mode mode;

**notify_next_wait3_func()**

BSD

Notify_value

notify_next_wait3_func(client, pid, status, rusage)
  Notify_client     client;
  int              pid;
  union wait *status;
  struct rusage   *rusage;

**notify_next_wait3_func()**

SYSV based systems

Notify_value

notify_next_wait3_func(client, pid, status, rusage)
  Notify_client     client;
  int              pid;
  int              *status;
  struct rusage   *rusage;
**notify_no_dispatch()**

Prevents the Notifier from dispatching events from within the call to `read` (2) or `select` (2).

```c
Notify_error
notify_no_dispatch()
```

See Also: `notify_do_dispatch()`.

**notify_perror()**

Analogous to the UNIX `perror` (3) system call. `s` is printed to `stderr`, followed by a terse description of `notify_errno()`.

```c
void
notify_perror(s)
char *s;
```

**notify_post_event()**

Posts a client event. A client event may be posted to the Notifier at any time. The poster of a client event may suggest to the Notifier when to deliver the event, but this is only a hint. The Notifier will see to it that it is delivered at an appropriate time (more on this below).

```c
typedef char * Notify_event;

Notify_error
notify_post_event(client, event, when_hint)
    Notify_client client;
    Notify_event event;
    Notify_event_type when_hint;
```

The client handle from `notify_set_event_func()` is passed to `notify_post_event()`. `event` is defined and interpreted solely by the client. `when_hint` is `NOTIFY_SAFE` or `NOTIFY_IMMEDIATE`. A return code of `NOTIFY_OK` indicates that the notification has been posted. Other values indicate an error condition. `NOTIFY_UNKNOWN_CLIENT` indicates that `client` is unknown to the Notifier. `NOTIFY_NO_CONDITION` indicates that `client` has no client event handler registered with the Notifier.

**notify_post_event_and_arg()**

Passes additional data with an event. For example, when the scrollbar posts an event to its owner to do a scroll, the scrollbar’s handle is passed as an argument along with the event. The function `notify_post_event_and_arg()` provides the argument-passing mechanism.
notify_post_event_and_arg(client, event, when_hint, arg,
    copy_func, release_func)

    Notify_client client;
    Notify_event event;
    Notify_event_type when_hint;
    Notify_arg arg;
    Notify_copy copy_func;
    Notify_release release_func;

    typedef caddr_t Notify_arg;
    typedef Notify_arg (*Notify_copy)();
    #define NOTIFY_COPY_NULL ((Notify_copy)0)

    typedef void (*Notify_release)();
    #define NOTIFY_RELEASE_NULL ((Notify_release)0)

copy_func() is called to copy arg (and, optionally, event) when event and
arg need to be queued for later delivery. release_func() is called to release
the storage allocated during the copy call when event and arg were no longer needed by
the Notifier.

Any of arg, copy_func(), or release_func() may be NULL. If copy_func
is not NOTIFYCOPY.NULL and arg is NULL, then copy_func() is called anyway.
This allows event the opportunity to be copied because copy_func() takes a
pointer to event. The event pointed to may be replaced as a side effect of the copy
call. The same applies to a NOTIFY_RELEASE_NULL release function with a NULL
arg argument.

The copy() and release() routines are client-dependent, so you must write them
yourself. Their calling sequences are the following:

    Notify_arg
    copy_func(client, arg, event_ptr)
        Notify_client client;
        Notify_arg arg;
        Notify_event *event_ptr;

    void
    release_func(client, arg, event)
        Notify_client client;
        Notify_arg arg;
        Notify_event event;

notify_remove_destroy_func()
notify_remove_event_func()

Notify_error
notify_remove_event_func(client, event_func, when)
    Notify_client client;
    Notify_func event_func;
    int when;

notify_remove_exception_func()

Notify_error
notify_remove_exception_func(client, exception_func, fd)
    Notify_client client;
    Notify_func exception_func;
    int fd;

notify_remove_input_func()

Notify_error
notify_remove_input_func(client, input_func, fd)
    Notify_client client;
    Notify_func input_func;
    int fd;

notify_remove_itimer_func()

Notify_error
notify_remove_itimer_func(client, itimer_func, which)
    Notify_client client;
    Notify_func itimer_func;
    int which;

notify_remove_output_func()

Notify_error
notify_remove_output_func(client, output_func, fd)
    Notify_client client;
    Notify_func output_func;
    int fd;

notify_remove_signal_func()

Notify_error
notify_remove_signal_func(client, signal_func, signal, mode)
    Notify_client client;
    Notify_func signal_func;
    int signal;
    Notify_signal_mode mode;

notify_remove_wait3_func()

Notify_error
notify_remove_wait3_func(client, pid)
    Notify_client client;
    Notify_func wait3_func;
    int pid;
**notify_set_destroy_func()**

Registers `destroy_func()` with the Notifier. `destroy_func()` is called when a destroy event is posted to `client` or when the process receives a `SIGTERM` signal.

```c
Notify_func
notify_set_destroy_func(client, destroy_func)
  Notify_client client;
  Notify_func destroy_func;
```

The format for the destroy function is:

```c
Notify_value
  destroy_func(client, status)
  Notify_client client;
  Destroy_status status;
```

**notify_set_event_func()**

Registers the event handler `event_func()` with the Notifier.

```c
Notify_error
notify_set_event_func(client, event_func, type)
  Notify_client client;
  Notify_func event_func;
  Notify_event_type type;
```

`type` is either `NOTIFY_SAFE` or `NOTIFY_IMMEDIATE`.

The format for the event function is:

```c
Notify_value
  event_func(client, event, arg, type)
  Notify_client client;
  Notify_event event;
  Notify_arg arg;
  Notify_event_type type;
```

**notify_set_exception_func()**

Registers the exception handler `exception_func()` with the Notifier. The only known devices that generate exceptions at this time are stream-based socket connections when an out-of-band byte is available.

```c
Notify_func
notify_set_exception_func(client, exception_func, fd)
  Notify_client client;
  Notify_func exception_func;
  int fd;
```

The format for the exception function is:

```c
Notify_value
  exception_func(client, fd)
  Notify_client client;
  int fd;
```
**notify_set_input_func()**

Registers `input_func()` with the Notifier. `input_func()` will be called whenever there is input pending on `fd`.

```c
Notify_func
notify_set_input_func(client, input_func, fd)
    Notify_client client;
    Notify_func input_func;
    int fd;
```

The format for the input function is:

```c
Notify_value
exception_func(client, fd)
    Notify_client client;
    int fd;
```

**notify_set_itimer_func()**

Registers the timeout event handler `itimer_func()` with the Notifier.

```c
Notify_func
notify_set_itimer_func(client, itimer_func, which, value, ovalue)
    Notify_client client;
    Notify_func itimer_func;
    int which;
    struct itimerval *value, *ovalue;
```

The semantics of `which`, `value`, and `ovalue` parallel the arguments to `setitimer(2)`. `which` is either `ITIMER_REAL` or `ITIMER_VIRTUAL`.

The format for the itimer function is:

```c
Notify_value
itimer_func(client, which)
    Notify_client client;
    int which;
```

**notify_set_output_func()**

Registers `output_func()` with the Notifier. `output_func()` will be called whenever output has been completed on `fd`.

```c
Notify_func
notify_set_output_func(client, output_func, fd)
    Notify_client client;
    Notify_func output_func;
    int fd;
```

The format for the output function is:

```c
Notify_value
exception_func(client, fd)
    Notify_client client;
    int fd;
```
**notify_set_signal_func()**

Registers the signal event handler `signal_func()` with the Notifier. `signal_func()` will be called whenever signal is caught by the Notifier. `when` can be either NOTIFY_SYNC or NOTIFY_ASYNC.

Calling `notify_set_signal_func()` with a NULL in the place of the `signal_func()` turns off checking for that signal for that client.

```c
Notify_func
notify_set_signal_func(client, signal_func, signal, when)
Notify_client client;
Notify_func signal_func;
int signal;
Notify_signal_mode when;
```

The format for the `signal_func` function is:

```c
Notify_value
signal_func(client, sig, mode)
Notify_client client;
int sig;
int mode;
```

**notify_set_wait3_func()**

Registers the function `wait3_func()` with the Notifier. The registered function is called after client's process identified by `pid` dies. To do the minimum processing, register the predefined function `notify_default_wait3()`.

```c
Notify_func
notify_set_wait3_func(client, wait3_func, pid)
Notify_client client;
Notify_func wait3_func;
int pid;
```

The format for the `wait3` function on BSD-based systems is:

```c
Notify_value
wait3_func(client, pid, status, rusage)
Notify_client client;
int pid;
union wait *status;
struct rusage *rusage;
```

The format for the `wait3` function on SYSV-based systems is:

```c
Notify_value
wait3_func(client, pid, status, rusage)
Notify_client client;
int pid;
int *status;
struct rusage *rusage;
```

**notify_start()**

Begins dispatching of events by the Notifier.

```c
Notify_error
notify_start()
```
notify_stop()
Terminates dispatching of events by the Notifier.

notify_veto_destroy()
Called from within a destroy event handler when status is DESTROY_CHECKING and
the application does not want to be destroyed.

OPENWIN_EACH_VIEW()
Macro providing built-in support for iteration across all the views contained in a given
openwin. Allows you to perform operations on multiple views for which there are no
openwin attributes.

OPENWIN_END_EACH
Closes the loop started by the macro OPENWIN_EACH_VIEW(). These macros are
meant to be used together, as in the following example:

OPENWIN_EACH_VIEW(openwin, view)
  Openwin openwin;
  Openwin_item item;
  xv_set(openwin, attributes, 0);
  OPENWIN_END_EACH;

panel_advance_caret()
Advances the input focus to the next item that can accept input focus. If on the last in-
put focus, rotate back to the first. Returns the new caret item or NULL if there are no
input focus items.

panel_backup_caret()
Backs the caret up to the previous input focus item. If already on the first input focus
item, rotate back to the last. Returns the new caret item or NULL if there are no input
focus items.

PANEL_CHECK_BOX
Macro for “PANEL_TOGGLE, PANEL_FEEDBACK, PANEL_MARK.” Creates non-
exclusive choice item(s) with check_boxes instead of boxes.

xv_create(panel, PANEL_CHECK_BOX, NULL);
PANEL_CHECK_BOX expands to:

   PANEL_TOGGLE,
   PANEL_FEEDBACK,  PANEL_MARK

To use an ATTR_LIST argument, the ATTR_LIST must be the first attribute in an attribute-value list. See PANEL_CHOICE_STACK for an example.

PANEL_CHOICE_STACK
Macro for "PANEL_CHOICE, PANEL_DISPLAY_LEVEL, PANEL_CURRENT." Creates an OPEN LOOK abbreviated choice menu button.

   xv_create(panel, PANEL_CHOICE_STACK, NULL);

PANEL_CHOICE_STACK expands to:

   PANEL_CHOICE,
   PANEL_DISPLAY_LEVEL,  PANEL_CURRENT

To use an ATTR_LIST argument, the ATTR_LIST must be the first attribute in an attribute-value list. You need to include an explicit PANEL_CHOICE, rather than the PANEL_CHOICE_STACK attribute. For example,

   xv_create(owner, PANEL_CHOICE,
            ATTR_LIST, at,
            PANEL_DISPLAY_LEVEL,PANEL_CURRENT,
            .....,
            NULL);

PANEL_EACH_ITEM()
Macro to iterate over each item in a panel. The corresponding macro PANEL_END_EACH closes the loop opened by PANEL_EACH_ITEM().

   PANEL_EACH_ITEM(panel, item)
   Panel panel;
   Panel_item item;

PANEL_END_EACH
Closes the loop started by PANEL_EACH_ITEM(). Same usage as OPENWIN_EACH_VIEW().

panel_paint()
Paints an item or an entire panel. paint_behavior can be either PANEL_CLEAR, which causes the area occupied by the panel or item to be cleared prior to painting, or PANEL_NO_CLEAR.

   int panel_paint(panel_object, paint_behavior)
   Xv_object   panel_object;
   Panel_setting paint_behavior;

Note that panel_object may be a PANEL or a PANEL_ITEM. If panel_object does not exist, or panel_paint() is called with an invalid paint_behavior value, the function returns XV_ERROR; otherwise, it returns XV_OK.
**panel_text_notify()**
Default notify procedure for panel text items. Determines what the correct Panel_setting value should be based on the event passed into the notify procedure. The Panel_setting value causes the PANEL_TEXT package to adjust the caret to advance on Return or Tab, caret to back up on Shift-Return or Shift-Tab, printable characters to be inserted into item’s value, and all other characters to be discarded.

```c
Panel_setting
panel_text_notify(item, event)
    Panel_item    item
    Event         *event

Returns PANEL_NEXT, PANEL_PREVIOUS, PANEL_INSERT, or PANEL_NONE, respectively.
```

**PANEL_TOGGLE**
Macro for “PANEL_CHOICE, PANEL_CHOOSE_ONE, FALSE.” Used to create non-exclusive choice item(s).

```c
xv_create(panel, PANEL_TOGGLE, NULL);
```

PANEL_TOGGLE expands to:

```c
PANEL_CHOICE,
PANEL_CHOOSE_ONE, FALSE
```
To use an ATTR_LIST argument, the ATTR_LIST must be the first attribute in an attribute-value list. See PANEL_CHOICE_STACK for an example.

**rect_below()**
Returns TRUE if r2 lies below r1 AND the left edge of r2 isn’t greater than the right edge of r1 AND the right edge isn’t less than the left edge of r1.

```c
int rect_below( r1, r2 )
    Rect *r1, *r2;
```

**rect_borderadjust()**
Used to adjust the borders. This macro takes a rect pointer and an int. This macro is defined as follows:

```c
#define rect_borderadjust(r,m) \
{ (r)->r_width+=(m)+(m);(r)->r_height+=(m)+(m);}
```

**rect_bottom()**
Takes a rect pointer and returns the position of the bottom of the rect. This macro is defined as follows:

```c
#define rect_bottom(rect) ((rect)->r_top+(rect)->r_height-1)
```
rect_bounding()

Return the rectangle that defines the region bound by rectangle r1 and r2.

```
struct rect
    rect_bounding( r1, r2 )
    struct rect *r1, *r2;
```

rect_clipvector()

Clip vector defined by coordinates x1y1, x2y2 to space within rectangle r; returns TRUE if original vector resides within rectangle.

```
unsigned
    rect_clipvector( r, x1, y1, x2, y2 )
    struct rect *r;
    int x1, y1, x2, y2;
```

rect_construct()

Constructs a rect based on the values specified. It takes a rect pointer and four int arguments. This macro is defined as follows:

```
#define rect_construct(r,x,y,w,h) 
    { (r)->r_left=(x); (r)->r_top=(y); (r)->r_width=(w); (r)->r_height=(h); } 
```

rect_distance()

Compute the distance from coordinate xy to rectangle. If xy is inside rectangle, 0 is returned. If x_used or y_used are non-zero, then the projection point is returned.

```
int
    rect_distance( rect, x, y, x_used, y_used )
    Rect *rect;
    int x, y;
    int x_used, y_used;
```

rect_equal()

Compares two rect pointers and returns TRUE if all dimensions are equal. This macro is defined as follows:

```
#define rect_equal(r1,r2) 
    ( (r1)->r_left==(r2)->r_left && (r1)->r_width==(r2)->r_width && 
    (r1)->r_top==(r2)->r_top && (r1)->r_height==(r2)->r_height ) 
```

rect_includespoint()

Returns TRUE if specified coordinates are within specified rect. It takes a rect pointer and two integers (x and y coordinates). This macro is defined as follows:

```
#define rect_includespoint(r,x,y) 
    ( (x) >= (r)->r_left && (y) >= (r)->r_top && 
    (x)<(r)->r_left+(r)->r_width && (y)<(r)->r_top+(r)->r_height ) 
```
**rect_includesrect()**
Determines whether or not a specified rect is contained in another. It takes two rect pointers. This macro is defined as follows:

```c
#define rect_includesrect(r1, r2) \
   ((r1)->r_left <= (r2)->r_left && \ 
    (r1)->r_top <= (r2)->r_top && \ 
    (r1)->r_left+(r1)->r_width >= (r2)->r_left+(r2)->r_width && \ 
    (r1)->r_top+(r1)->r_height >= (r2)->r_top+(r2)->r_height)
```

**rect_intersection()**
Calculates the intersection of rectangles r1 and r2 and returns the resulting rectangle r.

```c
struct rect
    rect_intersection( r1, r2, r )
    Rect *r1, *r2, *r;
```

**rect_intersectsrect()**
Determines whether or not one rect intersects another. It takes two rect pointers. This macro is defined as follows:

```c
#define rect_intersectsrect(r1,r2) \
   ((r1)->r_left<(r2)->r_left+(r2)->r_width && \ 
    (r1)->r_top<(r2)->r_top+(r2)->r_height && \ 
    (r2)->r_left<(r1)->r_left+(r1)->r_width && \ 
    (r2)->r_top<(r1)->r_top+(r1)->r_height)
```

**rect_isnull()**
Takes a rect pointer and returns TRUE if either the width or the height of the rect is 0. Otherwise returns FALSE. This macro is defined as follows:

```c
#define rect_isnull(r) ((r)->r_width == 0 || (r)->r_height == 0)
```

**rect_marginadjust()**
Adjusts the margins in a rect. It takes a rect pointer and an integer. This macro is defined as follows:

```c
#define rect_marginadjust(r,m) \
   { (r)->r_left-=(m);(r)->r_top-=(m); \ 
     (r)->r_width+=(m)+(m);(r)->r_height+=(m)+(m);}
```
rect_order()
Return TRUE if rectangles r1 and r2 are in specified sort order. Where sort order is defined by:

RECTS_TOPTOBOTTOM: returns TRUE if r1 top <= r2 top
RECTS_BOTTOMTOTOP: returns TRUE if r1 bottom >= r2 bottom
RECTS_LEFTORIGHT: returns TRUE if r1 left <= r2 left
RECTS_RIGHTTOLEFT: returns TRUE if r1 right >= r2 right
RECTS_UNSORT: returns TRUE

unsigned rect_order( r1, r2, sortorder )
struct rect *r1, *r2;
int sortorder;
rect_passtochild()
Takes two integers (x and y coordinates) and a rect pointer. This macro is defined as follows:

#define rect_passtochild(x,y,rect) 
{ (rect)->r_left=(rect)->r_left-(x); (rect)->r_top=(rect)->r_top-(y); }
rect_passtoparent()
Takes two integers (x and y coordinates) and a rect pointer. This macro is defined as follows:

#define rect_passtoparent(x,y,rect) 
{ (rect)->r_left=(rect)->r_left+(x); (rect)->r_top=(rect)->r_top+(y); }
rect_print()
Takes a rect pointer and prints the rectangle on stderr.

#define rect_print(rect) 
\( (void)fprintf(stderr,"[left: %d, top: %d, width: %d, height: %d]0, 
\ (rect)->r_left, (rect)->r_top, (rect)->r_width, (rect)->r_height)
rect_right()
Takes a rect pointer and returns the position of the right edge of the rect. This macro is defined as follows:

#define rect_right(rect) ((rect)->r_left+(rect)->r_width-1)
rect_right_of()
Returns TRUE if r2 lies to the right of r1 AND the bottom of r2 isn’t above the top of r1 AND the top of r2 isn’t below the bottom of r1.

int rect_right_of( r1, r2 )
Rect *r1, *r2;
rect_sizes_differ()

Takes two rect pointers. If all dimensions are equal, it returns FALSE, if not it returns TRUE. This macro is defined as follows:

```c
#define rect_sizes_differ(r1, r2) \
   ((r1)->r_width != (r2)->r_width || (r1)->r_height != (r2)->r_height)
```

SCROLLABLE_PANEL

Used to create a scrollbar panel. To add a scrollbar after creating the panel, create a scrollbar with the panel as its parent. For example:

```c
panel = xv_create(frame, SCROLLABLE_PANEL, NULL);
xv_create(panel, SCROLLBAR, NULL);
```

Note: Scrollable panels are not inherently OPEN LOOK-compliant.

scrollbar_paint()

Repaints all portions of the scrollbar.

```c
void scrollbar_paint(scrollbar)
   Scrollbar scrollbar;
```

selection_*

Appendix B, Selection Compatibility Procedures and Macros, describes the selection_* procedures. The selection_* procedures provide compatibility for applications using selections created prior to XView Version 3. If you are creating a new application, refer to Chapter 18, Selections, in the XView Programming Manual.

sel_convert_proc()

Default selection convert procedure provided by XView Version 3 for use with the SELECTION_OWNER package. This procedure allows the selection owner to communicate with the selection requestor.

```c
int sel_convert_proc(sel_owner, type, data, length, format)
   Selection_owner sel_owner;
   Atom *type; /* Input/Output */
   Xv_opaque *data;
   unsigned long *length; /* Output */
   int *format;
```

sel_owner is the selection owner. type is the form data should be converted to. data is a pointer to the reply buffer. length is a pointer to the length of the reply buffer. format is a pointer to an integer representing the number of bits in a single data member in reply buffer. Returns TRUE if completed successfully, FALSE if it does not complete successfully.
sel_post_req()

This procedure is used to send a non-blocking request to the selection owner. The application’s reply procedure is called by the SELECTION package when the selection owner has sent its reply.

```c
int sel_post_req( sel )
    Selection_requestor sel;
```

`sel` is the selection requestor. Returns `XV_OK` if it successfully sends the request, otherwise it returns `XV_ERROR`. When no reply procedure is defined, `XV_ERROR` is returned (see the description for the attribute `SEL_REPLY_PROC`).

textsw_add_mark()

Adds a new mark at position. `flags` can be either `TEXTSW_MARK_DEFAULTS` or `TEXTSW_MARK_MOVE_AT_INSERT`.

```c
Textsw_mark textsw_add_mark(textsw, position, flags)
    Textsw textsw;
    Textsw_index position;
    unsigned flags;
```

textsw_append_file_name()

Returns 0 if `textsw` is editing a file and, if so, appends the name of the file at the end of name.

```c
int textsw_append_file_name(textsw, name)
    Textsw textsw;
    char *name;
```

textsw_delete()

Returns 0 if the operation fails. Removes the span of characters beginning with `first` and ending one before `last_plus_one`.

```c
Textsw_index textsw_delete(textsw, first, last_plus_one)
    Textsw textsw;
    Textsw_index first, last_plus_one;
```

textsw_edit()

Returns 0 if the operation fails. Erases a character, a word or a line, depending on whether `unit` is `SELN_LEVEL_FIRST` or `SELN_LEVEL_LINE`. If `direction` is 0, characters after the insertion point are affected; otherwise, characters before the insertion point are affected. The operation will be done `count` times.

```c
Textsw_index textsw_edit(textsw, unit, count, direction)
    Textsw textsw;
    unsigned unit, count, direction;
```
textsw_erase()

Returns 0 if the operation fails. Equivalent to textsw_delete() but does not affect the global shelf.

```c
Textsw_index textsw_erase(textsw, first, last_plus_one)
Textsw   textsw;
Textsw_index first, last_plus_one;
```

textsw_file_lines_visible()

Fills in top and bottom with the file line indices of the first and last file lines being displayed in textsw.

```c
void
  textsw_file_lines_visible(textsw, top, bottom)
Textsw   textsw;
int       *top, *bottom;
```

textsw_find_bytes()

Beginning at the position addressed by first, searches for the pattern specified by buf of length buf_len. Searches forward if flags is 0, else searches backward. Returns -1 if no match, else matching span placed in indices addressed by first and last_plus_one.

```c
int
  textsw_find_bytes(
    textsw, first, last_plus_one, buf, buf_len, flags)
Textsw   textsw;
Textsw_index  *first,  *last_plus_one;
char       *buf;
unsigned    buf_len, flags;
```

textsw_find_mark()

Returns the current position of mark. If this operation fails, it will return TEXTSW_INFINITY.

```c
Textsw_index textsw_find_mark(textsw, mark)
Textsw   textsw;
Textsw_mark  mark;
```

textsw_first()

Returns the first textsw view.

```c
Textsw
  textsw_first(textsw)
Textsw   textsw;
```
textsw_index_for_file_line()

Returns the character index for the first character in the line given by line. If this operation fails, it will return TEXTSW_CANNOT_SET.

Textsw_index
textsw_index_for_file_line(textsw, line)
    Textsw  textsw;
    int     line;

textsw_insert()

Inserts characters in buf into textsw at the current insertion point. The number of characters actually inserted is returned. This will equal buf_len unless there was a memory allocation failure. If there was a failure, it returns NULL.

Textsw_insert
textsw_insert(textsw, buf, buf_len)
    Textsw  textsw;
    char    *buf;
    int     buf_len;

textsw_match_bytes()

Searches for a block of text in the textsw’s contents; ends with characters matching end_sym. This function places the starting index of the matching block in first and its ending index in last.

int
textsw_match_bytes(
    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;
    char    *start_sym, *end_sym;
    int     start_sym_len, end_sym_len;
    unsigned  field_flag;

textsw_next()

Returns the next view in the set of textsw views.

Textsw
textsw_next(textsw)
    Textsw  textsw;

textswNormalize_view()

Repositions the text so that the character at position is visible and at the top of the subwindow.

void
textsw_normalize_view(textsw, position)
    Textsw  textsw;
    Textsw_index  position;
textsw_possibly_normalize()

If the character at position is already visible, this function does nothing. If it is not visible, it repositions the text so that it is visible and at the top of the subwindow.

\[
\text{void}
\]
\[
\textsw_{\text{possibly\_normalize}}(\textsw, \ \text{position})
\]
\[
\textsw \ \textsw;
\]
\[
\textsw_{\text{index}} \ \text{position};
\]

textsw_remove_mark()

Removes an existing mark from textsw.

\[
\text{void}
\]
\[
\textsw_{\text{remove\_mark}}(\textsw, \ \text{mark})
\]
\[
\textsw \ \textsw;
\]
\[
\textsw_{\text{mark}} \ \text{mark};
\]

textsw_replace_bytes()

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

\[
\text{Textsw\_index}
\]
\[
\textsw_{\text{replace\_bytes}}(\textsw, \ \text{first}, \ \text{last\_plus\_one}, \ \text{buf}, \ \text{buf\_len})
\]
\[
\textsw \ \textsw;
\]
\[
\textsw_{\text{index}} \ \text{first};
\]
\[
\textsw_{\text{index}} \ \text{last\_plus\_one};
\]
\[
\text{char} \ \text{buf};;
\]
\[
\text{unsigned} \ \text{buf\_len};
\]

textsw_reset()

Discards edits performed on the contents of textsw. If needed, a message box will be displayed at x, y.

\[
\text{void}
\]
\[
\textsw_{\text{reset}}(\textsw, \ \text{x}, \ \text{y})
\]
\[
\textsw \ \textsw;
\]
\[
\text{x}, \ \text{y};
\]

textsw_save()

Saves any edits made to the file currently loaded into textsw. If needed, a message box will be displayed at x, y.

\[
\text{unsigned}
\]
\[
\textsw_{\text{save}}(\textsw, \ \text{x}, \ \text{y})
\]
\[
\textsw \ \textsw;
\]
\[
\text{x}, \ \text{y};
\]

208 XView Reference Manual
textsw_screen_line_count()

Returns the number of screen lines in textsw.

```c
int
textsw_screen_line_count(textsw)
    Textsw    textsw;
```

textsw_scroll_lines()

Moves the text up or down by count lines. If count is positive, then the text is scrolled up on the screen. If negative, the text is scrolled down (backward in the file).

```c
void
textsw_scroll_lines(textsw, count)
    Textsw    textsw;
    int       count;
```

textsw_set_selection()

Sets the selection to begin at first and includes all characters up to last_plus_one. A type value of 1 indicates primary selection, 2 indicates secondary selection.

```c
void
textsw_set_selection(textsw, first, last_plus_one, type)
    Textsw    textsw;
    Textsw_index first, last_plus_one;
    unsigned   type;
```

textsw_store_file()

Stores the contents of textsw to the file named by filename. If needed, a message box will be displayed at x, y.

```c
unsigned
textsw_store_file(textsw, filename, x, y)
    Textsw    textsw;
    char *filename;
    int       x, y;
```

ttysw_input()

Appends len number of characters from buf onto tty’s input queue. It returns the number of characters accepted.

```c
int
    ttysw_input(tty, buf, len)
        Tty     tty;
        char *buf;
        int    len;
```
ttysw_output()
Appends len number of characters from buf onto tty's output queue; it sends them through the terminal emulator to the TTY. It returns the number of characters accepted.

```c
int ttysw_output(tty, buf, len)
    Typ tty;
    char *buf;
    int len;
```

window_done()
Destroys the entire hierarchy to which win belongs.

```c
int window_done(win)
    Xv_Window win;
```

window_fit()
This macro causes win to fit its contents in the dimensions specified with WIN_FIT_HEIGHT and WIN_FIT_WIDTH. It is defined as:

```c
xv_set(win, WIN_FIT_HEIGHT, 0, WIN_FIT_WIDTH, 0, NULL)
```

window_fit_height()
Causes win to fit its contents in the height specified with WIN_FIT_HEIGHT. The macro is defined as:

```c
xv_set(win, WIN_FIT_HEIGHT, 0, NULL)
```

window_fit_width()
Causes win to fit its contents in the width specified with WIN_FIT_WIDTH. The macro is defined as:

```c
xv_set(win, WIN_FIT_WIDTH, 0, NULL)
```

window_read_event()
Reads the next input event for window. In case of error, sets the global variable errno and returns -1.

```c
int window_read_event(window, event)
    Xv_Window window;
    Event *event;
```

win_set_kbd_focus
Assigns input focus (via XSetInputFocus()) to the passed window.

```c
int win_set_kbd_focus(window, xid)
    Xv_Window window; /* XView window to set focus on */
    XID xid; /* XID of XView window */
```
wmgr_bottom()
This procedure is a SunView Compatibility procedure. Sets stackmode to Below in XConfigureWindow.

```c
void
wmgr_bottom(frame)
    Frame frame;
```

wmgr_changelevel()
This procedure is a SunView Compatibility procedure. It sets the stackmode in XConfigureWindow.

```c
void
wmgr_changelevel(window, parent, top)
    Xv_Window window;
    int parent;
    int top;
```

wmgr_close()
Sets wmhints.initial_state to ICONICSTATE.

```c
void
wmgr_close(frame)
    Frame frame;
```

wmgr_completechangerect()
Calls XConfigureWindow with a new rect.

```c
void
wmgr_completechangerect(window, rectnew, rectoriginal, parentpleft, parentprtop)
    Xv_window window;
    Rect *rectnew, *rectoriginal;
    int parentpleft, parentprtop;
```

wmgr_open()
Sets wmhints.initial_state to NORMALSTATE.

```c
void
wmgr_open(frame)
    Frame frame;
```

wmgr_refreshwindow()
Calls XConfigureWindow repeatedly.

```c
void
wmgr_refreshwindow(window)
    Xv_window window;
```
wmgr_top()  
Sets stackmode to Above in XConfigureWindow.

```c
void
wmgr_top(frame)
    Frame  frame;
```

xv_col()  
Returns an integer representing the number of pixels, excluding the left margin of the window. This may be used in conjunction with a panel's WIN_COL_GAG.

```c
int
xv_col(window, column)
    Xv_Window  window;
    int         column;
```

xv_cols()  
Returns an integer representing the number of pixels, including the left margin of the window.

```c
int
xv_cols(window, columns)
    Xv_Window  window;
    int         columns;
```

xv_create()  
To create any XView object, call the generic procedure xv_create. This procedure will return a handle to some XView object. It takes as parameters the owner of the object being created, the type of object to create, and a list of attributes. The attribute list must terminate with a NULL. In specifying the type, you must use the name of some XView package. That name must be in all capital letters to distinguish it from the corresponding data type. The package is the name of the package to which the object you wish to create belongs.

The procedure xv_create() returns either the handle for the new object or XV_NULL if the attempt at object creation fails.

```c
XvOpaque
xv_create(owner, package, attributes)
    Xv_Object  owner;
    Xv_Pkg     *package;
    Attribute-List  attributes;
```

xv_destroy()  
To destroy an XView object and any subframes owned by that object, use the procedure xv_destroy(). It will return either XV_ERROR or XV_OK.

```c
int
xv_destroy(object)
    Xv_Object  object;
```
XV_DISPLAY_FROM_WINDOW()
Macro to get at the handle of the display structure from a window object.

XvOpaque
XV_DISPLAY_FROM_WINDOW(window)
Xv_Window window;

xv_error()
Is called by the XView internals in the event of an error.

char *
xv_error(object, attributes)
XvOpaque object;
attribute-list attributes;

xv_error_format()
This function calls a pointer to a static char * describing the XView error that has occurred. It should be copied into your buffer if you wish to retain the value since subsequent calls overwrite the contents.

char *
xv_error_format(object, avlist)
XvObject object;
Attr_avlist avlist;

xv_find()
To find any XView object, use the procedure xv_find(). If the object is not found, xv_find will automatically attempt to create it.

XvOpaque
xv_find(owner, package, attributes)
XvObject owner;
Xv_PKG *package;
attribute-list attributes;

xv_get()
To get the value of any single attribute of any XView object, use the procedure xv_get().

XvOpaque
xv_get(object, attribute)
XvObject object;
attribute-list attributes;

The procedure returns 0 for failure. As a result, you cannot detect errors when retrieving the values of attributes which might return 0 as a valid value. Note that, although you can supply XV_NULL as the owner when you create a new object with xv_create, you must provide the object when asking for the value of an attribute.
For example, you cannot just ask for:

```c
xv_get(XV_NULL, XV_FONT)
```

to determine the default font, but you can ask for the font associated with a particular screen, as in:

```c
xv_get(Screen1, XV_FONT)
```

Some attributes require a screen or server to be supplied to `xv_get()`. For such attributes, you should ask any window you think should be on the same screen what to identify as its screen or server, using the attributes `XV_SCREEN` and `XV_SERVER`.

If you know your application only runs on a single screen, you can use the global values `xv_default_server` and `xv_default_screen` or you can ask the defaults database what the server is.

**xv_init()**

This procedure performs many tasks including: opening the connection to the server, initializing the Notifier, and initializing the Resource Manager database (see Chapter 17, *Resources*, in the *XView Programming Manual*). By default, `xv_init()` opens a connection to the server described by the `DISPLAY` environment variable. The server connections may be changed using appropriate command-line attribute-value pairs. No matter which server is used, `xv_init()` returns a handle to that server object. For further information concerning use with servers, see Chapter 15, *Nonvisual Objects*, in the *XView Programming Manual*.

```c
int main(argc, argv)
    int argc;
    char **argv;
{
    /* initialization/declarations */
    ...
    (void)xv_init(XV_INIT_ARGC_PTR_ARGV, &argc, argv, 0);
    frame = xv_create(NULL, FRAME, FRAME_LABEL, "foo", 0);
}
```

**xv_row()**

Returns an integer representing the number of pixels, excluding the top margin of the window. Used in conjunction with a panel’s `WIN_ROW_GAP`.

```c
int xv_row(window, row)
    Xv_Window window
    int row;
```
XV_ROWS()
Returns an integer representing the number of pixels, including the top margin of the window.

```c
int xv_rows(window, rows)
    Xv_Window window
    int   rows;
```

XV_SCREEN_FROM_WINDOW()
Macro to return the handle to the screen object from the window object.

```c
Xv_screen
XV_SCREEN_FROM_WINDOW(window)
    Xv_Window window;
```

XV_SEND_MESSAGE()
Lets two separate processes communicate with each other. You can specify the addressee field either with one of the constants XV_POINTER_WINDOW or XV_FOCUS_WINDOW or with the window’s XID, if known. If the addressee is XV_POINTER, then the message is sent to the window that the pointer is in. If the addressee is XV_FOCUS_WINDOW, then the message is sent to the window that currently has the focus, regardless of the pointer position. If the addressee is a window’s XID, then the message is sent to the window with the corresponding ID.

```c
void xv_send_message(
    window, addressee, msg_type, format, data, len)
    Xv_object    window;
    Xv_opaque    addressee;
    char *msg_type;
    int   format;
    Xv_opaque *data;
    int   len;
```

msg_type is a pointer to a string specifying the property you want the Atom for. format is the number of bits of a single data member (8, 16, or 32). data is a pointer to the data to send. len is the number of bytes of data to send.

XV_SERVER_FROM_WINDOW()
Macro returns the handle to the server object from a window object.

```c
Xv_opaque
XV_SERVER_FROM_WINDOW(window)
    Xv_Window    window;
```
**xv_set()**

To set the value of one or more attributes of any XView object, call the procedure `xv_set` with the handle to the object whose attributes you wish to set and a list of attribute-value pairs terminating in a **NULL**.

```c
Xv_object
Xv_set(owner, type, attributes)
     Xv_object  owner;
     vu type    *type;
     attribute-list attributes;
```

The procedure `xv_set()` returns **XV_OK** if it succeeds; otherwise, it returns an error code indicating that the attribute on which it ran encountered problems.

**xv_super_set_avlist()**

This function handles the parsing of attributes that are generic to a package’s super set.

```c
Xv_public Xv_opaque
xv_super_set_avlist(object, pkg, avlist)
     register Xv_opaque  object;
     register Xv_pkg    *pkg;
     Attr_avlist         avlist;
```

**xv_unique_key()**

Generates a key for use with **XV_KEY_DATA**.

**Warning:** The uniqueness of any particular key is not guaranteed since the value returned is generated simply by incrementing a pre-defined “unique” value.

```c
int
xv_unique_key()
```

**xv_usage()**

Prints a list of toolkit’s generic command-line arguments.

```c
void
xv_usage(app_name)
     char *app_name;
```

See Also: **XV_USAGE_PROC**

**xv_window_loop()**

The function `xv_window_loop()` maps the frame passed as an argument and makes other frames and windows in the application “deaf.” This is similar to **FRAME_BUSY** except that the cursor does not change to a stopwatch and the frame header does not show the gray “busy” pattern. `xv_window_loop()` does not lock the screen.

```c
Xv_opaque
xv_window_loop(frame)
     Frame frame;
```

The frame passed into `xv_window_loop()` can have more than one subwindow of any type (this was a restriction with SunView’s `window_loop()` function).
xv_window_loop() does not return until a call to xv_window_return().

void
xv_window_return(return_val)
Xv_opaque return_val;

This presumably occurs in a callback that originates from a button (for example) on the frame. The return value passed into xv_window_return() is the value returned by xv_window_loop(). Since the screen is not locked when xv_window_loop() is active, the user might be able to dismiss the frame using the window manager. Doing this causes the application to hang since xv_window_return() is not called. To avoid this behavior, you need to attach a destroy procedure to the frame. The destroy procedure should call xv_window_return().

xv_window_return()
See the procedure xv_window_loop().
This page intentionally left blank

to preserve original page counts.
This page intentionally left blank
to preserve original page counts.
## Data Types

The following is a list of XView data types and their descriptions.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Canvas</strong></td>
<td>Handle to an opaque structure that describes a canvas.</td>
</tr>
<tr>
<td><strong>Canvas_attribute</strong></td>
<td>One of the canvas attributes (CANVAS_*)</td>
</tr>
<tr>
<td><strong>Xv_Cursor</strong></td>
<td>Handle to an opaque structure that describes a cursor.</td>
</tr>
<tr>
<td><strong>Xv_Cursor_attribute</strong></td>
<td>One of the cursor attributes (CURSOR_*)</td>
</tr>
<tr>
<td><strong>Destroy_status</strong></td>
<td>Enumeration:</td>
</tr>
<tr>
<td></td>
<td>- DESTROY_PROCESS_DEATH</td>
</tr>
<tr>
<td></td>
<td>- DESTROY_CHECKING</td>
</tr>
<tr>
<td></td>
<td>- DESTROY_CLEANUP</td>
</tr>
<tr>
<td></td>
<td>- DESTROY_SAVE_YOURSELF</td>
</tr>
<tr>
<td><strong>Dnd</strong></td>
<td>Handle to an opaque structure that describes a drag and drop object.</td>
</tr>
<tr>
<td><strong>DndDragType</strong></td>
<td>Enumeration:</td>
</tr>
<tr>
<td></td>
<td>- DND_MOVE=0</td>
</tr>
<tr>
<td></td>
<td>- DND_COPY</td>
</tr>
<tr>
<td><strong>Xv_drag_drop</strong></td>
<td>Handle to an opaque structure that describes a drag and drop object.</td>
</tr>
<tr>
<td><strong>Drop_site_item</strong></td>
<td>Handle to an opaque structure that describes a drop site.</td>
</tr>
<tr>
<td><strong>Xv_drop_site</strong></td>
<td>Handle to an opaque structure that describes a drop site.</td>
</tr>
<tr>
<td><strong>Event</strong></td>
<td>The structure that describes an input event. Use macros for access:</td>
</tr>
</tbody>
</table>
|                             | ```
|                             |     typedef struct inputevent {
|                             |         short ie_code;
|                             |             /* input code */
|                             |         short ie_flags;
|                             |         short ie_shiftmask;
|                             |             /* input code shift state */
|                             |         short ie_locx, ie_locy;
|                             |             /* locator (usually a mouse) position */
|                             |     };
|                             | ```
struct timeval ie_time;
    /* time of event */
short action;
    /* keymapped version of ie_code */
Xv_object ie_win;
    /* window the event is directed to */
char *ie_string;
    /* String returned from XLookupString
       * or language translation string. */
XEvent *ie_xevent;
    /* pointer to actual XEvent struct */
} Event;

Xv_Font
     Pointer to an opaque structure that describes a font.
Xv_Font_attribute
     One of the font attributes (FONT_*).
Frame
     Pointer to an opaque structure that describes a frame.
Frame_attribute
     One of the frame attributes (FRAME_*).
Fullscreen
     Handle to an opaque structure that describes a fullscreen.
Fullscreen_attribute
     One of the fullscreen attributes (FULLSCREEN_*).
Icon
     Handle to an opaque structure that describes an icon.
Icon_attribute
     One of the icon attributes (ICON_*).
Inputmask
     Mask specifying which input events a window will receive.
Menu
     Pointer to an opaque structure that describes a menu.
Menu_attribute
     One of the menu attributes (MENU_*).
Menu_generate
     Enumerated type for the operation parameter passed to generate
     procs.  Enumeration:
     MENU_DISPLAY
     MENU_DISPLAY_DONE
     MENU_NOTIFY
     MENU_NOTIFY_DONE
Xv_notice
     Handle to an opaque structure that describes a notice.
Xv_Notice
     Handle to an opaque structure that describes a notice.
Menu_item
     Pointer to an opaque structure that describes a menu item.
Notice_attribute
     One of the notice attributes (NOTICE_*).
Notify_arg
     Opaque client optional argument.
Notify_destroy
     Enumeration:
     NOTIFY_SAFE
     NOTIFY_IMMEDIATE
Notify_event
     Opaque client event.
Notify_event_type

Enumeration of errors for notifier functions:
- NOTIFY_SAFE
- NOTIFY_IMMEDIATE

Notify_func

Notifier function.

Notify_signal_mode

Enumeration:
- NOTIFY_SYNC
- NOTIFY_ASYNC

Notify_value

Enumeration of possible return values for client notify proc:
- NOTIFY_DONE
- NOTIFY_IGNORED
- NOTIFY_UNEXPECTED

Openwin_split_direction

Enumeration:
- OPENWIN_SPLIT_HORIZONTAL
- OPENWIN_SPLIT_VERTICAL

Panel

Pointer to an opaque structure that describes a panel.

Panel_attr

One of the panel attributes (PANEL_*).

Panel_item

Pointer to an opaque structure that describes a panel item.

Panel_item_type

Enumerated type:
- PANEL_ABBREV_MENU_BUTTON_ITEM
- PANEL_BUTTON_ITEM
- PANEL_CHOICE_ITEM
- PANEL_DROP_TARGET_ITEM
- PANEL_EXTENSION_ITEM
- PANEL_GAUGE_ITEM
- PANEL_ITEM
- PANEL_LIST_ITEM
- PANEL_MESSAGE_ITEM
- PANEL_MULTILINE_TEXT_ITEM
- PANEL_NUMERIC_TEXT_ITEM
- PANEL_SLIDER_ITEM
- PANEL_TEXT_ITEM
- PANEL_TOGGLE_ITEM

Panel_list_op

Enumerated type:
- PANEL_LIST_OP_DELETE
- PANEL_LIST_OP_DESELECT
- PANEL_LIST_OP_SELECT
- PANEL_LIST_OP_VALIDATE

Data Types
Panel_setting

Enumerated type:

- PANEL_CLEAR
- PANEL_NO_CLEAR
- PANEL_NONE
- PANEL_ALL
- PANEL_NON_PRINTABLE
- PANEL_SPECIFIED
- PANEL_CURRENT
- PANEL_DONE
- PANEL_MARKED
- PANEL_VERTICAL
- PANEL_HORIZONTAL
- PANEL_INVERTED
- PANEL_INSERT
- PANEL_NEXT
- PANEL_PREVIOUS
- PANEL_NONE_DOWN
- PANEL_LEFT_DOWN
- PANEL_MIDDLE_DOWN
- PANEL_RIGHT_DOWN
- PANEL_CHORD_DOWN

Rect

The structure describing a rectangle:

```c
typedef struct rect {
    short r_left;
    short r_top;
    short r_width;
    short r_height;
} Rect;
```

Rectlist

A list of rectangles:

```c
typedef struct rectlist {
    short rl_x, rl_y;
    Rectnode *rl_head;
    Rectnode *rl_tail;
    Rect rl_bound;
} Rectlist;
```

Rectnode

One of the individual rectangles in a rectlist:

```c
typedef struct rectnode {
    Rectnode *rn_next;
    Rect rn_rect;
} Rectnode;
```

Xv_Screen

Pointer to an opaque structure that describes a screen.

Xv_Screen_attr

One of the screen attributes (SCREEN_*).

Scrollbar

The opaque handle for a scrollbar.

Scrollbar_attr

One of the scrollbar attributes (SCROLL_*).
Scrollbar_motion

Enumeration:
- SCROLL_ABSOLUTE
- SCROLL_POINT_TO_MIN
- SCROLL_PAGE_FORWARD
- SCROLL_LINE_FORWARD
- SCROLL_MIN_TO_POINT
- SCROLL_PAGE_BACKWARD
- SCROLL_LINE_BACKWARD
- SCROLL_TO_END
- SCROLL_TO_START
- SCROLL_NONE

Scrollbar_setting

Enumeration:
- SCROLL_VERTICAL
- SCROLL_HORIZONTAL

Selection_item
Handle to an opaque structure that describes the selection item.

Selection_owner
Handle to an opaque structure that describes the selection owner.

Selection_requestor
Handle to an opaque structure that describes the selection requestor.

Seln_attribute
One of the seln attributes.

Xv_Server
Pointer to an opaque structure that describes a server.

Server_attr
One of the server attributes.

Server_image
Pointer to an opaque structure that describes a server image.

Server_image_attribute
One of the Server_image attributes:
- SERVER_IMAGE_DEPTH
- SERVER_IMAGE_BITS

Xv_single_color
Color values for a pixel. Defined in <xview/cms.h>.

typedef struct xv_singlecolor {
    u_char red, green, blue;
} Xv_singlecolor;

Textsw
Pointer to an opaque structure that describes a text subwindow.
Textsw_action | Enumeration of actions defined for client provided
| notify_proc:
| TEXTSW_ACTION_CAPS_LOCK
| TEXTSW_ACTION_CHANGED_DIRECTORY
| TEXTSW_ACTION_EDITED_FILE
| TEXTSW_ACTION_EDITED_MEMORY
| TEXTSW_ACTION_FILE_IS_READONLY
| TEXTSW_ACTION_LOADED_FILE
| TEXTSW_ACTION_TOOL_CLOSE
| TEXTSW_ACTION_TOOL_DESTROY
| TEXTSW_ACTION_TOOL_QUIT
| TEXTSW_ACTION_TOOL_MGR
| TEXTSW_ACTION_USING_MEMORY

Textsw_attribute | One of the Textsw attributes.

Textsw_enum | Miscellaneous Textsw Enumerations:
| TEXTSW_NEVER
| TEXTSW_ALWAYS
| TEXTSW_ONLY
| TEXTSW_IF_AUTO_SCROLL
| TEXTSW_CLIP
| TEXTSW_WRAP_AT_CHAR
| TEXTSW_WRAP_AT_WORD
| TEXTSW_WRAP_AT_LINE

Textsw_index | An index for a character within a text subwindow’s text stream.

Textsw_view | Pointer to an opaque structure that describes a text subwindow view.

Textsw_status | Enumeration describing the status of textsw.build and textsw.init.

Tty | Pointer to an opaque structure that describes a TTY subwindow.

Xv_Window | Pointer to an opaque structure that describes a window.

Window_attr | One of the window attributes (WIN_*)

Window_scale_state | Enumeration:
| WIN_SCALE_SMALL
| WIN_SCALE_MEDIUM
| WIN_SCALE_LARGE
| WIN_SCALE_EXTRALARGE
Xv_error_action

Enumeration:
XV_ERROR_CONTINUE
XV_ERROR_RETRY
XV_ERROR_ABORT

Xv_error_attr

Enumeration:
XV_ERROR_SYSTEM
XV_ERROR_BAD_VALUE
XV_ERROR_CREATE_ONLY
XV_ERROR_CANNOT_SET
XV_ERROR_CANNOT_GET
XV_ERROR_SERVER
XV_ERROR_STRING
XV_ERROR_INVALID_OBJ
XV_ERROR_INTERNAL

Xv_error_severity

Enumeration:
XV_ERROR_RECOVERABLE
XV_ERROR_NON_RECOVERABLE

Xv_object

Pointer to an opaque structure that describes an XView object.

Xv_opaque

Pointer to an opaque structure.

Xv_xrectlist

A list of rectangles returned by XView in X11 rectangle list format. Defined in xv_xrect.h

#define XV_MAX_XRECTS 32
typedef struct {
    XRectangle rect_array[XV_MAX_XRECTS];
    int count;
} Xv_xrectlist;
This page intentionally left blank

to preserve original page counts.
Table 5-1 lists the predefined event codes and their values. Table 5-2 lists the event codes for the Mouseless Model, and their values.

### Table 5-1. Event Codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Event Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ASCII_FIRST</td>
<td>Marks beginning of ASCII range.</td>
</tr>
<tr>
<td>127</td>
<td>ASCII_LAST</td>
<td>Marks end of ASCII range.</td>
</tr>
<tr>
<td>128</td>
<td>META_FIRST</td>
<td>Marks beginning of META range.</td>
</tr>
<tr>
<td>255</td>
<td>META_LAST</td>
<td>Marks end of META range.</td>
</tr>
<tr>
<td>31744</td>
<td>ACTION_NULL_EVENT</td>
<td>Event was not translated into an action.</td>
</tr>
<tr>
<td>31745</td>
<td>ACTION_ERASE_CHAR_BACKWARD</td>
<td>Erase char to the left of caret.</td>
</tr>
<tr>
<td>31746</td>
<td>ACTION_ERASE_CHAR_FORWARD</td>
<td>Erase char to the right of caret.</td>
</tr>
<tr>
<td>31747</td>
<td>ACTION_ERASE_WORD_BACKWARD</td>
<td>Erase word to the left of caret.</td>
</tr>
<tr>
<td>31748</td>
<td>ACTION_ERASE_WORD_FORWARD</td>
<td>Erase word to the right of caret.</td>
</tr>
<tr>
<td>31749</td>
<td>ACTION_ERASE_LINE_BACKWARD</td>
<td>Erase to the beginning of the line.</td>
</tr>
<tr>
<td>31750</td>
<td>ACTION_ERASE_LINE_END</td>
<td>Erase to the end of the line.</td>
</tr>
<tr>
<td>31752</td>
<td>ACTION_GO_CHAR_BACKWARD</td>
<td>Move the caret one character to the left.</td>
</tr>
<tr>
<td>31753</td>
<td>ACTION_GO_CHAR_FORWARD</td>
<td>Move the caret one character to the right.</td>
</tr>
<tr>
<td>31754</td>
<td>ACTION_GO_WORD_BACKWARD</td>
<td>Move the caret one word to the left.</td>
</tr>
<tr>
<td>31755</td>
<td>ACTION_GO_WORD_FORWARD</td>
<td>Move the caret one word to the right.</td>
</tr>
<tr>
<td>31756</td>
<td>ACTION_GO_WORD_END</td>
<td>Move the caret to the end of the word.</td>
</tr>
<tr>
<td>31757</td>
<td>ACTION_GO_LINE_BACKWARD</td>
<td>Move the caret to the start of the line.</td>
</tr>
<tr>
<td>31758</td>
<td>ACTION_GO_LINE_FORWARD</td>
<td>Move the caret to the start of the next line.</td>
</tr>
<tr>
<td>31759</td>
<td>ACTION_GO_LINE_END</td>
<td>Move the caret to the end of the line.</td>
</tr>
<tr>
<td>31760</td>
<td>ACTION_GO_LINE_START</td>
<td>Move the caret to the beginning of the line.</td>
</tr>
<tr>
<td>31761</td>
<td>ACTION_GO_COLUMN_BACKWARD</td>
<td>Move the caret up one line, maintaining column position.</td>
</tr>
<tr>
<td>31762</td>
<td>ACTION_GO_COLUMN_FORWARD</td>
<td>Move the caret down one line, maintaining column position.</td>
</tr>
<tr>
<td>31763</td>
<td>ACTION_GO_DOCUMENT_START</td>
<td>Move the caret to the beginning of the text.</td>
</tr>
<tr>
<td>31764</td>
<td>ACTION_GO_DOCUMENT_END</td>
<td>Move the caret to the end of the text.</td>
</tr>
<tr>
<td>31765</td>
<td>ACTION_GO_PAGE_FORWARD</td>
<td>Move the caret to the next page.</td>
</tr>
<tr>
<td>31766</td>
<td>ACTION_GO_PAGE_BACKWARD</td>
<td>Move the caret to the previous page.</td>
</tr>
<tr>
<td>31767</td>
<td>ACTION_STOP</td>
<td>Stop the operation.</td>
</tr>
<tr>
<td>Value</td>
<td>Event Code</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>31768</td>
<td>ACTION_AGAIN</td>
<td>Repeat previous operation.</td>
</tr>
<tr>
<td>31769</td>
<td>ACTION_PROPS</td>
<td>Show property sheet window.</td>
</tr>
<tr>
<td>31770</td>
<td>ACTION_UNDO</td>
<td>Undo previous operation.</td>
</tr>
<tr>
<td>31771</td>
<td>ACTION_REDO</td>
<td>Repeat previous operation.</td>
</tr>
<tr>
<td>31772</td>
<td>ACTION_FRONT</td>
<td>Bring window to the front of the desktop.</td>
</tr>
<tr>
<td>31773</td>
<td>ACTION_BACK</td>
<td>Put the window at the back of the desktop.</td>
</tr>
<tr>
<td>31774</td>
<td>ACTION_COPY</td>
<td>Copy the selection to the clipboard.</td>
</tr>
<tr>
<td>31775</td>
<td>ACTION_OPEN</td>
<td>Open a window from its icon form (or close if already open).</td>
</tr>
<tr>
<td>31776</td>
<td>ACTION_CLOSE</td>
<td>Close a window to an icon.</td>
</tr>
<tr>
<td>31777</td>
<td>ACTION_PASTE</td>
<td>Copy clipboard contents to the insertion point.</td>
</tr>
<tr>
<td>31778</td>
<td>ACTION_FIND_BACKWARD</td>
<td>Find the text selection to the left of the caret.</td>
</tr>
<tr>
<td>31779</td>
<td>ACTION_FIND_FORWARD</td>
<td>Find the text selection to the right of the caret.</td>
</tr>
<tr>
<td>31780</td>
<td>ACTION_REPLACE</td>
<td>Show find and replace window.</td>
</tr>
<tr>
<td>31781</td>
<td>ACTION_CUT</td>
<td>Delete the selection and put on clipboard.</td>
</tr>
<tr>
<td>31782</td>
<td>ACTION_SELECT_FIELD_BACKWARD</td>
<td>Select the previous delimited field.</td>
</tr>
<tr>
<td>31783</td>
<td>ACTION_SELECT_FIELD_FORWARD</td>
<td>Select the next delimited field.</td>
</tr>
<tr>
<td>31784</td>
<td>ACTION_COPY_THEN_PASTE</td>
<td>Copy, then paste, text.</td>
</tr>
<tr>
<td>31785</td>
<td>ACTION_STORE</td>
<td>Store the specified selection as a new file.</td>
</tr>
<tr>
<td>31786</td>
<td>ACTION_LOAD</td>
<td>Load the specified selection as a new file.</td>
</tr>
<tr>
<td>31787</td>
<td>ACTION_INCLUDE_FILE</td>
<td>Includes the file.</td>
</tr>
<tr>
<td>31788</td>
<td>ACTION_GET_FILENAME</td>
<td>Get the selected filename.</td>
</tr>
<tr>
<td>31789</td>
<td>ACTION_GET_DIRECTORY</td>
<td>Set the directory to the selection.</td>
</tr>
<tr>
<td>31790</td>
<td>ACTION_DO_IT</td>
<td>Do the appropriate default action.</td>
</tr>
<tr>
<td>31791</td>
<td>ACTION_HELP</td>
<td>Set the directory to the selection.</td>
</tr>
<tr>
<td>31792</td>
<td>ACTION_INSERT</td>
<td>&quot;INSERT&quot; key. This may not be available on all keyboards.</td>
</tr>
<tr>
<td>31796</td>
<td>ACTION_CAPS_LOCK</td>
<td>Toggle caps-lock state.</td>
</tr>
<tr>
<td>31799</td>
<td>ACTION_SELECT</td>
<td>Left mouse button down or up.</td>
</tr>
<tr>
<td>31800</td>
<td>ACTION_ADJUST</td>
<td>Middle mouse button down or up.</td>
</tr>
<tr>
<td>31801</td>
<td>ACTION_MENU</td>
<td>Right mouse button down or up.</td>
</tr>
<tr>
<td>31802</td>
<td>ACTION_DRAG_MOVE</td>
<td>For moving text.</td>
</tr>
<tr>
<td>31803</td>
<td>ACTION_DRAG_COPY</td>
<td>Attempting to drag copy.</td>
</tr>
<tr>
<td>31803</td>
<td>ACTION_SPLIT_HORIZONTAL</td>
<td>Split pane horizontally.</td>
</tr>
<tr>
<td>31804</td>
<td>ACTION_DRAG_LOAD</td>
<td>Attempting to drag load.</td>
</tr>
<tr>
<td>31806</td>
<td>ACTION_SPLIT_VERTICAL</td>
<td>Split pane vertically.</td>
</tr>
<tr>
<td>31807</td>
<td>ACTION_SPLIT_INIT</td>
<td>Initialize a split pane.</td>
</tr>
<tr>
<td>31808</td>
<td>ACTION_SPLITDESTROY</td>
<td>Destroy a split of a pane.</td>
</tr>
<tr>
<td>31809</td>
<td>ACTION_RESIZE</td>
<td>Rescale a pane.</td>
</tr>
<tr>
<td>31810</td>
<td>ACTION_PININ</td>
<td>Pop up’s OPEN LOOK pushpin in window header is in.</td>
</tr>
<tr>
<td>31811</td>
<td>ACTION_PINOUT</td>
<td>Pop up’s OPEN LOOK pushpin in window header is out.</td>
</tr>
<tr>
<td>31812</td>
<td>ACTION_DISMISS</td>
<td>OPEN LOOK “dismiss” of pop-up window.</td>
</tr>
<tr>
<td>31815</td>
<td>ACTION_TAKE_FOCUS</td>
<td>Take the input focus.</td>
</tr>
<tr>
<td>Value</td>
<td>Event Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>----------------------------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td>31818</td>
<td>KBD_MAP</td>
<td>KeymapNotify</td>
</tr>
<tr>
<td>31819</td>
<td>WIN_GRAPHICS_EXPOSE</td>
<td>GraphicsExpose</td>
</tr>
<tr>
<td>31820</td>
<td>WIN_NO_EXPOSE</td>
<td>NoExpose</td>
</tr>
<tr>
<td>31821</td>
<td>WIN_VISIBILITY_NOTIFY</td>
<td>VisibilityNotify</td>
</tr>
<tr>
<td>31822</td>
<td>WIN_CREATE_NOTIFY</td>
<td>CreateNotify</td>
</tr>
<tr>
<td>31823</td>
<td>WIN_DESTROY_NOTIFY</td>
<td>DestroyNotify</td>
</tr>
<tr>
<td>31824</td>
<td>WIN_MAP_REQUEST</td>
<td>MapRequest</td>
</tr>
<tr>
<td>31825</td>
<td>WIN_REPAARENT_NOTIFY</td>
<td>ReparentNotify</td>
</tr>
<tr>
<td>31826</td>
<td>WIN_GRAVITY_NOTIFY</td>
<td>GravityNotify</td>
</tr>
<tr>
<td>31827</td>
<td>WIN_RESIZE_REQUEST</td>
<td>ResizeRequest</td>
</tr>
<tr>
<td>31828</td>
<td>WIN_CONFIGURE_REQUEST</td>
<td>ConfigureRequest</td>
</tr>
<tr>
<td>31829</td>
<td>WIN_CIRCULATE_REQUEST</td>
<td>CirculateRequest</td>
</tr>
<tr>
<td>31830</td>
<td>WIN_CIRCULATE_NOTIFY</td>
<td>CirculateNotify</td>
</tr>
<tr>
<td>31831</td>
<td>WIN_PROPERTY_NOTIFY</td>
<td>PropertyNotify</td>
</tr>
<tr>
<td>31832</td>
<td>WIN_COLORMAP_NOTIFY</td>
<td>ColormapNotify</td>
</tr>
<tr>
<td>31836</td>
<td>MAPPING_NOTIFY</td>
<td>MappingNotify</td>
</tr>
<tr>
<td>31895</td>
<td>ACTION_MATCH_DELIMITER</td>
<td>Select text up to a matching delimiter.</td>
</tr>
<tr>
<td>31897</td>
<td>ACTION_QUOTE</td>
<td>Cause next event in the input stream to pass. untranslated by the keymapping system.</td>
</tr>
<tr>
<td>31898</td>
<td>ACTION_EMPTY</td>
<td>Empty out the object or window.</td>
</tr>
<tr>
<td>32000</td>
<td>PANEL_EVENT_CANCEL</td>
<td>The panel or panel item is no longer “current.”</td>
</tr>
<tr>
<td>32256</td>
<td>SCROLLBAR_REQUEST</td>
<td>Request the scrollbar client to scroll paint window to a new view start.</td>
</tr>
<tr>
<td>32512</td>
<td>LOC_MOVE</td>
<td>MotionNotify – Pointer moves.</td>
</tr>
<tr>
<td>32513</td>
<td>LOC_WINENTER</td>
<td>EnterNotify – Pointer enters window.</td>
</tr>
<tr>
<td>32514</td>
<td>LOC_WINEXIT</td>
<td>LeaveNotify – Pointer exits window.</td>
</tr>
<tr>
<td>32515</td>
<td>LOC_DRAG</td>
<td>MotionNotify – Pointer moves while a button was down.</td>
</tr>
<tr>
<td>32516</td>
<td>WIN_REPAINT</td>
<td>Expose – Some portion of window requires repainting.</td>
</tr>
<tr>
<td>32517</td>
<td>WIN_RESIZE</td>
<td>ConfigureNotify – Window has been resized.</td>
</tr>
<tr>
<td>32518</td>
<td>WIN_MAP_NOTIFY</td>
<td>MapNotify – Notification of window being mapped.</td>
</tr>
<tr>
<td>32519</td>
<td>WIN_UNMAP_NOTIFY</td>
<td>UnmapNotify – Notification of window that is being unmapped.</td>
</tr>
<tr>
<td>32520</td>
<td>KBD_USE</td>
<td>FocusIn – Window is now the focus of keyboard input.</td>
</tr>
<tr>
<td>32521</td>
<td>KBD_DONE</td>
<td>FocusOut – Window is no longer the focus of input from keyboard.</td>
</tr>
<tr>
<td>32522</td>
<td>WIN_CLIENT_MESSAGE</td>
<td>ClientMessage – A message from another client.</td>
</tr>
<tr>
<td>32522</td>
<td>WIN_GRAPHICS_EXPOSE</td>
<td>GraphicsExpose – Source area for copy is outside of source window or obscured.</td>
</tr>
<tr>
<td>32537</td>
<td>SEL_CLEAR</td>
<td>SelectionClear</td>
</tr>
<tr>
<td>32538</td>
<td>SEL_REQUEST</td>
<td>SelectionRequest</td>
</tr>
<tr>
<td>32539</td>
<td>SEL_NOTIFY</td>
<td>SelectionNotify</td>
</tr>
<tr>
<td>32563+i-1</td>
<td>BUT (i)</td>
<td>Press pointer buttons 1–10</td>
</tr>
</tbody>
</table>
Table 5-1. Event Codes (continued)

<table>
<thead>
<tr>
<th>Value</th>
<th>Event Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>32563</td>
<td>MS_LEFT</td>
<td>Press left mouse button. ButtonPress or ButtonRelease.</td>
</tr>
<tr>
<td>32564</td>
<td>MS_MIDDLE</td>
<td>Press middle mouse button. ButtonPress or ButtonRelease.</td>
</tr>
<tr>
<td>32565</td>
<td>MS_RIGHT</td>
<td>Press right mouse button. ButtonPress or ButtonRelease.</td>
</tr>
<tr>
<td>32573+i-1</td>
<td>KEY_LEFT(i)</td>
<td>Press left function keys 1–15. KeyPress or KeyRelease.</td>
</tr>
<tr>
<td>32589+i-1</td>
<td>KEY_RIGHT(i)</td>
<td>Press right function keys 1–15. KeyPress or KeyRelease.</td>
</tr>
<tr>
<td>32605+i-1</td>
<td>KEY_TOP(i)</td>
<td>Press top function keys 1–15. KeyPress or KeyRelease.</td>
</tr>
<tr>
<td>3261+i-1</td>
<td>KEY_BOTTOM(i)</td>
<td>“BOTTOM” keys</td>
</tr>
<tr>
<td>3261</td>
<td>KEY_BOTTOMLEFT</td>
<td></td>
</tr>
<tr>
<td>3261</td>
<td>KEY_BOTTOMFIRST</td>
<td></td>
</tr>
<tr>
<td>3262</td>
<td>KEY_BOTTOMRIGHT</td>
<td></td>
</tr>
<tr>
<td>32636</td>
<td>KEY_BOTTOMLAST</td>
<td></td>
</tr>
</tbody>
</table>

Table 5-2. Mouseless Event Codes

<table>
<thead>
<tr>
<th>Value</th>
<th>Event Code</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>31818</td>
<td>ACTION_ACCELERATOR</td>
<td>XVIEW_FIRST+74</td>
</tr>
<tr>
<td>31819</td>
<td>ACTION_DELETE_SELECTION</td>
<td>XVIEW_FIRST+75</td>
</tr>
<tr>
<td>31820</td>
<td>ACTION_ERASE_LINE</td>
<td>XVIEW_FIRST+76</td>
</tr>
<tr>
<td>31821</td>
<td>ACTION_HORIZONTAL_SCROLLBAR_MENU</td>
<td>XVIEW_FIRST+77</td>
</tr>
<tr>
<td>31822</td>
<td>ACTION_INPUT_FOCUS_HELP</td>
<td>XVIEW_FIRST+78</td>
</tr>
<tr>
<td>31823</td>
<td>ACTION_JUMP_DOWN</td>
<td>XVIEW_FIRST+79</td>
</tr>
<tr>
<td>31824</td>
<td>ACTION_JUMP_MOUSE_TO_INPUT_FOCUS</td>
<td>XVIEW_FIRST+80</td>
</tr>
<tr>
<td>31825</td>
<td>ACTION_JUMP_UP</td>
<td>XVIEW_FIRST+81</td>
</tr>
<tr>
<td>31826</td>
<td>ACTION_MORE_HELP</td>
<td>XVIEW_FIRST+82</td>
</tr>
<tr>
<td>31827</td>
<td>ACTION_MORE_TEXT_HELP</td>
<td>XVIEW_FIRST+83</td>
</tr>
<tr>
<td>31828</td>
<td>ACTION_NEXT_ELEMENT</td>
<td>XVIEW_FIRST+84</td>
</tr>
<tr>
<td>31829</td>
<td>ACTION_NEXT_PANE</td>
<td>XVIEW_FIRST+85</td>
</tr>
<tr>
<td>31830</td>
<td>ACTIONPane_BACKGROUND</td>
<td>XVIEW_FIRST+86</td>
</tr>
<tr>
<td>31831</td>
<td>ACTIONPane_LEFT</td>
<td>XVIEW_FIRST+87</td>
</tr>
<tr>
<td>31832</td>
<td>ACTIONPane_RIGHT</td>
<td>XVIEW_FIRST+88</td>
</tr>
<tr>
<td>31833</td>
<td>ACTION_PANEL_START</td>
<td>XVIEW_FIRST+89</td>
</tr>
<tr>
<td>31834</td>
<td>ACTION_PANEL_END</td>
<td>XVIEW_FIRST+90</td>
</tr>
<tr>
<td>31835</td>
<td>ACTION_PREVIOUS_ELEMENT</td>
<td>XVIEW_FIRST+91</td>
</tr>
<tr>
<td>31836</td>
<td>ACTION_PREVIOUS_PANE</td>
<td>XVIEW_FIRST+92</td>
</tr>
<tr>
<td>31837</td>
<td>ACTION_QUOTE_NEXT_KEY</td>
<td>XVIEW_FIRST+93</td>
</tr>
<tr>
<td>Value</td>
<td>Event Code</td>
<td>Definition</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>31838</td>
<td>ACTION_RESUME_MOUSELESS</td>
<td>XVIEW_FIRST+94</td>
</tr>
<tr>
<td>31839</td>
<td>ACTION_SCROLL_DATA_END</td>
<td>XVIEW_FIRST+95</td>
</tr>
<tr>
<td>31840</td>
<td>ACTION_SCROLL_DATA_START</td>
<td>XVIEW_FIRST+96</td>
</tr>
<tr>
<td>31841</td>
<td>ACTION_SCROLL_DOWN</td>
<td>XVIEW_FIRST+97</td>
</tr>
<tr>
<td>31842</td>
<td>ACTION_SCROLL_JUMP_DOWN</td>
<td>XVIEW_FIRST+98</td>
</tr>
<tr>
<td>31843</td>
<td>ACTION_SCROLL_JUMP_LEFT</td>
<td>XVIEW_FIRST+99</td>
</tr>
<tr>
<td>31844</td>
<td>ACTION_SCROLL_JUMP_RIGHT</td>
<td>XVIEW_FIRST+100</td>
</tr>
<tr>
<td>31845</td>
<td>ACTION_SCROLL_JUMP_UP</td>
<td>XVIEW_FIRST+101</td>
</tr>
<tr>
<td>31846</td>
<td>ACTION_SCROLL_LEFT</td>
<td>XVIEW_FIRST+102</td>
</tr>
<tr>
<td>31847</td>
<td>ACTION_SCROLL_LINE_END</td>
<td>XVIEW_FIRST+103</td>
</tr>
<tr>
<td>31848</td>
<td>ACTION_SCROLL_LINE_START</td>
<td>XVIEW_FIRST+104</td>
</tr>
<tr>
<td>31849</td>
<td>ACTION_SCROLL_RIGHT</td>
<td>XVIEW_FIRST+105</td>
</tr>
<tr>
<td>31850</td>
<td>ACTION_SCROLL_PANE_DOWN</td>
<td>XVIEW_FIRST+106</td>
</tr>
<tr>
<td>31851</td>
<td>ACTION_SCROLL_PANE_LEFT</td>
<td>XVIEW_FIRST+107</td>
</tr>
<tr>
<td>31852</td>
<td>ACTION_SCROLL_PANE_RIGHT</td>
<td>XVIEW_FIRST+108</td>
</tr>
<tr>
<td>31853</td>
<td>ACTION_SCROLL_PANE_UP</td>
<td>XVIEW_FIRST+109</td>
</tr>
<tr>
<td>31854</td>
<td>ACTION_SCROLL_UP</td>
<td>XVIEW_FIRST+110</td>
</tr>
<tr>
<td>31855</td>
<td>ACTION_SELECT_ALL</td>
<td>XVIEW_FIRST+111</td>
</tr>
<tr>
<td>31856</td>
<td>ACTION_SELECT_DATA_END</td>
<td>XVIEW_FIRST+112</td>
</tr>
<tr>
<td>31857</td>
<td>ACTION_SELECT_DATA_START</td>
<td>XVIEW_FIRST+113</td>
</tr>
<tr>
<td>31858</td>
<td>ACTION_SELECT_DOWN</td>
<td>XVIEW_FIRST+114</td>
</tr>
<tr>
<td>31859</td>
<td>ACTION_SELECT_JUMP_DOWN</td>
<td>XVIEW_FIRST+115</td>
</tr>
<tr>
<td>31860</td>
<td>ACTION_SELECT_JUMP_LEFT</td>
<td>XVIEW_FIRST+116</td>
</tr>
<tr>
<td>31861</td>
<td>ACTION_SELECT_JUMP_RIGHT</td>
<td>XVIEW_FIRST+117</td>
</tr>
<tr>
<td>31862</td>
<td>ACTION_SELECT_JUMP_UP</td>
<td>XVIEW_FIRST+118</td>
</tr>
<tr>
<td>31863</td>
<td>ACTION_SELECT_LEFT</td>
<td>XVIEW_FIRST+119</td>
</tr>
<tr>
<td>31864</td>
<td>ACTION_SELECT_LINE_END</td>
<td>XVIEW_FIRST+120</td>
</tr>
<tr>
<td>31865</td>
<td>ACTION_SELECT_LINE_START</td>
<td>XVIEW_FIRST+121</td>
</tr>
<tr>
<td>31866</td>
<td>ACTION_SELECT_RIGHT</td>
<td>XVIEW_FIRST+122</td>
</tr>
<tr>
<td>31867</td>
<td>ACTION_SELECT_PANE_DOWN</td>
<td>XVIEW_FIRST+123</td>
</tr>
<tr>
<td>31868</td>
<td>ACTION_SELECT_PANE_LEFT</td>
<td>XVIEW_FIRST+124</td>
</tr>
<tr>
<td>31869</td>
<td>ACTION_SELECT_PANE_RIGHT</td>
<td>XVIEW_FIRST+125</td>
</tr>
<tr>
<td>31870</td>
<td>ACTION_SELECT_PANE_UP</td>
<td>XVIEW_FIRST+126</td>
</tr>
<tr>
<td>31871</td>
<td>ACTION_SELECT_UP</td>
<td>XVIEW_FIRST+127</td>
</tr>
<tr>
<td>31872</td>
<td>ACTION_SUSPEND_MOUSELESS</td>
<td>XVIEW_FIRST+128</td>
</tr>
<tr>
<td>31873</td>
<td>ACTION_TEXT_HELP</td>
<td>XVIEW_FIRST+129</td>
</tr>
<tr>
<td>31874</td>
<td>ACTION_TRANSLATE</td>
<td>XVIEW_FIRST+130</td>
</tr>
<tr>
<td>31875</td>
<td>ACTION_VERTICAL_SCROLLBAR_MENU</td>
<td>XVIEW_FIRST+131</td>
</tr>
</tbody>
</table>
This page intentionally left blank

to preserve original page counts.
This page intentionally left blank
to preserve original page counts.
This section lists XView properties that can be set with command-line options or by specifying values for resources. Values for most properties can be overridden programatically using XView attributes. For example, the position and size of an application’s base frame can be set using the command-line option \texttt{-geometry}, or by changing the value of the \texttt{Window.Geometry} resource. However, if the application sets its size and position by setting attributes with \texttt{xv_set()}, the values specified in the call to \texttt{xv_set()} take precedence.

This section is divided into two parts: 6.1, which lists the properties that have an explicit command-line option, and 6.2, which lists the properties that do not have an explicit command-line option.

XView properties may be set in a number of ways, as the following list indicates. The list shows precedence for setting property values, from highest to lowest.

1. Calling \texttt{xv\_set()} for one or more attributes that change the value of a property.
2. Specifying a flag name plus its value, if any, on the command line.
3. Editing or adding values for resource defaults in a \texttt{/Xdefaults} file.
4. Calling \texttt{xv\_create()} for an object's attributes.
5. Using the package default values for the property.

From this list, you can see that command-line options have precedence over values set during \texttt{xv\_create()}, but that all values are overridden by an explicit call to \texttt{xv\_set()}. The command-line options \texttt{-rv} (reverse), \texttt{-scale}, \texttt{-font}, \texttt{-foreground\_color}, \texttt{-background\_color}, and \texttt{-icon\_font} apply to all top-level frames in the application. The remaining options apply only to the first top-level frame created by the application. Child subframes inherit properties from their parent frames. Therefore, command-line options eventually propagate to subframes unless the options are overridden programatically.
6.1 Command-line Options with Resources

--background
This option takes a single argument that is in the form of a predefined color name (lavender, gray, goldenrod, etc.) from $OPENWINHOME/lib/rbg.txt or a hexadecimal representation. The hexadecimal representation is of the form pound sign (#) followed by the hexadecimal representation of the red, green, and blue aspects of the color. Also see -Wb.
Short: bg
Type: string (color name, or hexadecimal color specification)
Resource: Window.Color.Background
Default: white
Examples:

  cmdtool -foreground blue -background gray
Provides a blue foreground, with a gray background.

  cmdtool -foreground #d800ff -background white
Provides a purple foreground, with a white background.

--default
This option allows the user to set resources that don’t have command-line equivalents. The format is --default resource-name value. The XView resources without specific command-line arguments are discussed in the following section.
Short: Wd
Type: string string
Resource: Given by the first string
Default: None
Example:

  cmdtool -default
  OpenWindows.ScrollbarPlacement left

--defeateventsecurity
Enables an XView application to receive all synthethic events (sent via XSendEvent()).
Short: WS
Type: boolean
Resource: none
Default: FALSE
Example:

  cmdtool -WS
--depth
Specifies the window’s depth.
Type: int
Resource: Window.Depth

--disable_retained
This option is useful for applications running on a monochrome display, where server memory is at a minimum. For performance reasons, monochrome windows are by default retained by the server. Using retained windows will use more memory in the X11 server; however, it also speeds up repainting when the window is covered and uncovered by other windows. When --disable_retained is set, monochrome windows are not retained, thus saving server memory.

Short: -Wdr
Type: Boolean
Resource: Window.Mono.DisableRetained
Default: Not retained on color systems. Retained on monochrome systems.

--disable_xio_error_handler
This option is useful for debugging an application. Whenever there is a fatal XIO error, the server will print an error message before exiting. XView installs an error handler to keep those messages from appearing. If you would like to see these messages, use this option.

Short: -Wdxio
Type: Boolean
Resource: None
Default: Enable XIO handler. Setting this option disables the XIO handler.

--display
Sets the name of the X11 server on which to connect. Host is the name or address of the machine on whose server you have permission to display. Display is a number corresponding to the server on which to display for that machine, and screen corresponds to which screen for the server. See manual page on xhost for more details on adding to permissions list.

Short: -Wr
Type: String of the form: host:display{.screen}
Resource: Server.Name
Default: Taken from the DISPLAY environment variable.
Example:

cmdtool -display foobar:0
Brings up a cmdtool on the default screen of the display #0 on the host foobar.

cmdtool -display foobar:0.1
Brings up a cmdtool on screen #1 of display #0 of host foobar.
--font, -fn
Sets the name of the font used for the application (not control areas). To find out what fonts are available, use the `xlsfonts` command.

Short: -Wf
Type: string
Resource: Font.Name or OpenWindows.RegularFont
Default: lucidasans-12
Example:

```
  cmdtool -font fixed
```

If the font you specify cannot be found, an error message is shown.

For example:

```
XView warning: Cannot load font ‘galant-24’ (Font package)
XView warning: Attempting to load font ‘-
  bkh-lucida-medium-r-normal-sans-*-120-
  -*-*-_*-*-*-*’ instead (Font package)
```

--foreground
This option specifies the foreground color. For example, an application’s text in its textsw would take on the foreground color. Also see the --background option for information on similar functions. This option takes a single argument that is in the form of a predefined color name (lavender, gray, goldenrod, etc.) from $OPENWINHOME/lib/rbg.txt or a hexadecimal representation. The hexadecimal representation is of the form pound sign (#) followed by the hexadecimal representation of the red, green, and blue aspects of the color.

Short: -fg
Type: string
(resource name, or hexadecimal color specification)
Resource: Window.Color.Foreground
Default: black
Example:

```
  cmdtool -fg blue
```

Comes up with a blue foreground.

--foreground_color
This option allows the user to specify the foreground color of an application. It takes three values that should be integers between 0 and 255. They specify the amount of red, green, and blue that is in the color.

Short: -Wf
Type: integer integer integer
Resource: Window.Color.Foreground
Default: 0 0 0
Example:

```
  cmdtool -Wf 0 0 255
```

Comes up with a blue foreground.
--fullscreendebug
Enables or disables fullscreen debugging mode during which XGrabs (XGrabServer(), XGrabKeyboard(), XGrabPointer()) are not done. When using the FULLSCREEN package, the X11 server is normally grabbed. This prevents other windows on the server from responding until the grab has been released by the one window which initiated the grab. Refer to the manual Converting SunView Applications for further details.

Short: -Wfsdb
Type: Boolean
Resource: Fullscreen.Debug
Default: FALSE

--fullscreendebugkbd
Enables or disables keyboard grabbing using XGrabKeyboard() that is done via the FULLSCREEN package.

Short: -Wfsdbk
Type: Boolean
Resource: Fullscreen.Debugkbd
Default: FALSE

--fullscreendebugptr
Enables or disables pointer grabbing XGrabPointer() that is done via the FULLSCREEN package. Refer to Appendix F of the manual Converting SunView Applications for further details.

Short: -Wfsdbp
Type: Boolean
Resource: Fullscreen.Debugptr
Default: FALSE

--fullscreendebugserver
Enables or disables server grabbing using XGrabServer() that is done with the FULLSCREEN package. Refer to Appendix F in the manual Converting SunView Applications for further details.

Short: -Wfsdbs
Type: Boolean
Resource: Fullscreen.Debugserver
Default: FALSE

--geometry
This sets both the size and the placement of the application’s base frame. This option has priority over the -size and -position arguments. The size and placement parts of the value are optional. You can set just the size, just the position, or both. The size values are measured in pixels, and the position values use the same semantics as -position. However, if you use the – (minus) in front of an x value, it will be taken as relative to the right-hand side of the screen, instead of the left. Likewise, if you use the – (minus) with the y value, it will be taken relative to the bottom of the screen instead of the top.
Short: -WG
Type: string of the format: {WxH}{[+-]x[+-]y}
Resource: Window.Geometry
Default: Depends on the application.
Examples:

    cmdtool -geometry 500x600

Makes the base frame 500×600 pixels, with the position set by the window manager.

    cmdtool -WG +10+20

Makes the base frame of the default size with the left-hand side of the frame 10 pixels from the left-hand side of the screen, and the top of the frame 20 pixels from the top of the screen.

    cmdtool -WG -10+20

Makes the base frame of the default size with the right-hand side of the frame 10 pixels from the right-hand side of the screen, and the top of the frame 20 pixels from the top of the screen.

    cmdtool -geometry 400x300-0-0

Makes the base frame 400×300 pixels with the right-hand side of the frame flush against the right-hand side of the screen, and the bottom of the frame flush with the bottom of the screen.

−help

Prints a description of the valid command-line arguments for the application.

Short: -WH
Type: None
Resource: None
Default: None

−icon_font

Sets the name of the font used for the application’s icon. To find out what fonts are available, use xlsfonts.

Short: -WT
Type: string
Resource: Icon.Font.Name
Default: Depends on the application.
Example:

    cmdtool -WT "*century schoolbook"
-icon_image
Sets the default filename for the icon’s image. However, the application can overwrite this setting and display its own icon image. The file must be in XView icon format. The program iconedit allows you to create an image in the icon format. Several icons are available in the directory $OPENWINHOME/include/images. By convention, icon format files end with the suffix .icon.

Short: -WI
Type: string
Resource: Icon.Pixmap
Default: Depends on the application.
Example:

    cmdtool -WI /usr/include/images/stop.icon

-icon_label
Sets a default label for the base frame’s icon. However, the application can overwrite this setting and display its own icon label.

Short: -WL
Type: string
Resource: Icon.Footer
Default: Depends on the application.
Example:

    cmdtool -WL "Icon Label"

-icon_position
Sets the position of the application’s icon in pixels. Uses the same semantics as -position for base frames.

Short: -WP
Type: integer integer
Resource: Icon.X Icon.Y
Default: Depends on the window manager.
Example:

    cmdtool -WP 400 20

-label
Sets a default label for the base frame’s header. However, the application can overwrite this setting and display its own header.

Short: -Wl
Type: string
Resource: Window.Header
Default: Depends on the application.
Example:

    cmdtool -Wl "Header Text"
Locale setting is the method by which the language and cultural environment of a system is set. Locale setting affects the display and manipulation of language-dependent features.

The internationalization features that XView Version 3 supports include locale setting. One of the ways locale can be set is with command-line options. See the XView Programming Manual for details on other methods.

The \texttt{-lc\_basiclocale} option specifies the basic locale category, which sets the country of the user interface.

\begin{itemize}
\item Type: string
\item Resource: basicLocale
\item Default: “C”
\end{itemize}

\textbf{-lc\_displaylang}

Specifies the display language locale category, sets the language in which labels, messages, menu items, and help text are displayed.

\begin{itemize}
\item Type: string
\item Resource: displayLang
\item Default: “C”
\end{itemize}

\textbf{-lc\_inputlang}

Specifies the input language locale category, sets the language used for keyboard input.

\begin{itemize}
\item Type: string
\item Resource: inputLang
\item Default: “C”
\end{itemize}

\textbf{-lc\_numeric}

Specifies the numeric locale category, which defines the language used to format numeric quantities.

\begin{itemize}
\item Type: string
\item Resource: numeric
\item Default: “C”
\end{itemize}

\textbf{-lc\_timeformat}

Specifies the time format locale category, which defines the language used to format time and date.

\begin{itemize}
\item Type: string
\item Resource: timeFormat
\item Default: “C”
\end{itemize}
-name

Specifies the instance name of the application. This name is used to construct the resource name used to perform lookups in the X11 Resource Manager to look for the values for customizable attributes.

Type: string
Resource: None
Default: argv[0]

-position

Sets the initial position of the application's base frame in pixels. The upper left corner of the screen is at position (0,0) with the x-axis increasing to the left and the y-axis increasing downward. To determine framebuffer size, one can use the `eeprom` command on the local machine. To determine screen size for a remote display, one can use Xlib functions (see the Xlib Programming Manual). These values will also be generated by the “Save Workspace” option on the root menu into the $HOME/openwin-init file when using the Open Look Window Manager.

Short: -Wp
Resource: Window.X and Window.Y
Type: integer integer
Default: Depends on the window manager.
Example:

```
  cmdtool -Wp 100 200
```

-scale

Sets the initial scale of the application (larger or smaller). Small is 10 pixels, medium is 12 pixels, large is 14 pixels and extra_large is 19 pixels. Any font resource overrides the scale unless the scale is specified on the command line.

Short: -Wx
Type: string
Valid Values: {small, medium, large, extra_large}
Resource: Windows.Scale or OpenWindows.Scale
Default: “medium”
Example:

```
  cmdtool -scale extra_large
```

-size

Sets the width and height of the application's base frame. The values are in pixels.

Short: -Ws
Type: integer integer
Resource: Window.Width and Window.Height
Default: Depends on the application.
Example:

```
  cmdtool -Ws 400 500
```
**--synchronous, +synchronous**

These options allow you to make the connection that the application has with the X11 server either synchronous (-sync) or asynchronous (+sync).

- **Short:** -sync, +sync
- **Type:** Boolean
- **Resource:** Window.Synchronous
- **Default:** +synchronous

**--title**

Sets a default label for the base frame’s header. However, the application can overwrite this setting and display its own header.

- **Short:** -Wl
- **Type:** string
- **Resource:** Window.Header
- **Default:** Depends on the application

**Example:**

```
cmdtool -Wl "Header Text"
```

**--visual**

Resource specifies the visual used for the window.

- **Type:** Visual
- **Resource:** Window.visual

**--Wb**

This option allows the user to specify the background color (i.e., the color that text is painted on) for an application. The arguments are three values that should be integers between 0 and 255. They specify to the amount of red, green, and blue that is in the color.

- **Type:** integer integer integer
- **Resource:** Window.Color.Background
- **Default:** 0 0 0

**--Wd**

See the --default option.

**--Wdr**

See the --disable_retained option.

**--Wdxio**

See the --disable_xio_error_handler option.

**--Wf**

See the --foreground_color option.

**--Wfsdb**

See the --fullscreendebbug option.

**--Wfsdbs**

See the --fullscreendebbugserver option.
-WG  See the -geometry option.

-WH  See the -help option.

-Wi, and +Wi
These options control how an application will come up, open or closed (iconified).

Short:
Type:        Boolean
Resource:    Window.Iconic
Default:     +Wi
Examples:

    cmdtool +Wi
     Makes cmdtool come up open.
    cmdtool -Wi
     Makes cmdtool come up closed.

-WI  (uppercase i)
See the -icon_image option.

-WI  (lowercase L)
See the -label option.

-WL  See the -icon_label option.

-Wp  See the -position option.

-WP  See the -icon_position option.

-Wr  See the -display option.

-Ws  See the -size option.

-Wt  See the -font option.

-WT  See the -icon_font option.

-Wx  See the -scale option.
This option allows the user to set resources that don’t have command-line equivalents. This is similar to the –default option, but it takes only one argument, a string in the form of resource-name:value.

Type: String
Resource: Given in the string
Default: None
Example:

cmdtool -xrm OpenWindows.ScrollbarPlacement:right

6.2 Additional Resources

The .Xdefaults file stores resource settings. We recommend that you use the command-line arguments in order to change display characteristics rather than changing the .Xdefaults file. Changing the resources in .Xdefaults modifies the behavior of the user’s session. Novice users should not casually modify these settings.

Before attempting to edit the .Xdefaults file, please read the appropriate sections of the Xlib Programming Manual on the file format and the specific properties you intend to change.

Note that resources documented in this section do not have command-line arguments. It is possible to change these properties without altering the .Xdefaults file. The command-line arguments –xrm and –default provide instructions on how to specify values for any property. The resources that have command-line arguments are documented in the previous section.

Introduction.Resources (Props)

This is an example of the format for the resources described in this section. This field contains a brief description of the resource. If the resource can be modified by the OpenWindows Property Sheet, the resource name is followed by “(Props)”.

Values: Val1, Val2 (Default). This field contains the possible values for the resource. If the resource may contain any value, the default(s) are provided. Defaults are shown enclosed in parentheses following the valid values.

alarm.visible

When ringing the bell in an XView program, flash the window as well to warn the user.

Values: True, False (True)

keyboard.cancel

Values: integer
**keyboard.defaultAction**

Values: integer

**keyboard.inputFocusHelp**

Values: integer

**keyboard.nextElement**

Values: integer

**keyboard.deleteChar**

Specifies the delete character. This resource applies to text windows only and not to panel text items. This would work in either cmdtool or textedit or the compose window of mailtool.

Values: C (177 = octal for Delete)

Where C is some character either typed into an editor or specified with an octal equivalent.

**keyboard.deleteLine**

Specifies the delete line character. This resource applies to text windows only and not to panel text items. This would work in either cmdtool or textedit or the compose window of mailtool.

Values: C

Where C is some character either typed into an editor or specified with an octal equivalent.

**keyboard.deleteWord**

Specifies the delete word character. This resource applies to text windows only and not to panel text items. This would work in either cmdtool or textedit or the compose window of mailtool.

Values: C (27 = octal for \W)

Where C is some character either typed into an editor or specified with an octal equivalent.

**mouse.modifier.button2**

When using a mouse with less than three buttons, this resource gets an equivalent mapping for the second button which is the ADJUST button on a three button mouse. For more information on keysyms, see the xmodmap reference manual page in the Xlib documentation, or the include file $OPENWINHOME/include/X11/Xkeymap.h.

Values: Shift, Ctrl, any valid modifier keysym (Shift)

**mouse.modifier.button3**

When using a mouse with less than three buttons, this resource gets an equivalent mapping for the third button which is the MENU button on a three button mouse. For more information on keysyms, see the xmodmap reference manual page, Xlib documentation, and the include file $OPENWINHOME/include/X11/Xkeymap.h.

Values: Shift, Ctrl, any valid modifier keysym (Ctrl)
**mouse.multiclick.space**
Specifies the maximum number of pixels between successive mouse clicks to still have the clicks considered as a multi-click event.

Values:  
N (4)  
Where N is an integer between 2 and 500.

**notice.beepCount**
Ringing the bell can consist of either an audible beep and/or a visual flash.

Values:  
N (1)  
Where N is an integer to specify how many times to ring the bell when a notice appears.

**notice.jumpCursor**

Values:  True, False

**OpenWindows.3DLook.Monochrome**

Values:  True, False (True on all monochrome screens)

**OpenWindows.3DLook.Color**
When False, do not use the 3-D look on a color or grayscale screen.

Values:  True, False (True on all but monochrome screens)

**OpenWindows.beep (Props)**
When the value is notices, the audible bell will ring only when a notice pops up. When the value is never, the audible bell will never ring. When the value is always, the audible bell will always ring when the bell function is called by a program.

Values:  never, notices, always (always)

**OpenWindows.BoldFont**
Sets the font used by labels of a panel text item. If -font is specified on the command line, then it will override the font set by OpenWindows.BoldFont. To find out what fonts are available, use the xlsfonts command.

Values:  font name

**OpenWindows.dragRightDistance (Props)**
Used by menus to determine when a pullright submenu would display when dragging over the menu item near a submenu.

Values:  N (100)  
N is an integer greater than 0. A reasonable value might start at 20 and go to 200.
**OpenWindows.GotoMenu.UserDirs**

User’s directories in Goto Menu. `<string-list>` is a newline separated list of full-path directory names. Since the users home directory is always present it should not be in this list.

Values: `<string-list>`
Default: `<null-string>`
Example:

```
OpenWindows.GotoMenu.UserDirs: /home/me/src0usr/openwin
```

**OpenWindows.GotoMenu.RecentCount**

Number of Recently Visited Directories Shown: where `<count>` is an non-negative integer which is the number of the recently visited directories shown in the Goto Menu. The default is 8, the maximum is 15.

Values: `<count>`
Default: `<null-string>`
Example:

```
OpenWindows.GotoMenu.RecentCount: 10
```

**OpenWindows.KeyboardCommand.***

All of the `OpenWindows.KeyboardCommand` resource mappings may be modified by users, or by specifying one of three values for `OpenWindows.KeyboardCommands`: Sunview1 (which is the default), Basic, or Full. See the description for `OpenWindows.KeyboardCommands` and Chapter 6, *Handling Input*, in the *XView Programming Manual*, for more information.

**OpenWindows.KeyboardCommand.Adjust**

This mapping is loaded if `KeyboardCommands` is set to Full. This is a keyboard Core Functions resource.

Values: Insert+Alt

**OpenWindows.KeyboardCommand.Again**

This is a keyboard Core Functions resource.

Values: a+Meta, a+Ctrl+Meta, L2

**OpenWindows.KeyboardCommand.Copy**

This is keyboard Core Functions resource.

Values: u+Meta, L4

**OpenWindows.KeyboardCommand.CopyThenPaste**

This is keyboard Core Functions resource. This mapping is always loaded.

Values: p+Meta
OpenWindows.KeyboardCommand.Cut
This is keyboard Core Functions resource. This mapping is always loaded.
Values: x+Meta, L10

OpenWindows.KeyboardCommand.DataEnd
This mapping is always loaded. This is a keyboard Local Navigation command.
Default Values: End, R13, Return+Ctrl, End+Shift
Values (Basic or Full): End+Ctrl, R13+Ctrl

OpenWindows.KeyboardCommand.DataStart
Default Values: Home, R7, Return+Shift+Ctrl, Home+Shift
Values (Basic or Full): Home+Ctrl, R7+Ctrl

OpenWindows.KeyboardCommand.DefaultAction
This is Keyboard Core Functions resource. This mapping is always loaded.
Values: Return+Meta

OpenWindows.KeyboardCommand.Down
This mapping is always loaded. This is a keyboard Local Navigation command.
Default Values: n+Ctrl, P+Ctrl, Down, R14, Down+Shift
Values (Basic or Full): Down

OpenWindows.KeyboardCommand.Empty
This mapping is always loaded. This is a Text Editing command resource.
Values: e+Meta, e+Ctrl+Meta

OpenWindows.KeyboardCommand.EraseCharBackward
This mapping is always loaded. This is a Text Editing command resource.
Values: Delete, BackSpace

OpenWindows.KeyboardCommand.EraseCharForward
This mapping is always loaded. This is a Text Editing resource.
Values: Delete+Shift, BackSpace+Shift

OpenWindows.KeyboardCommand.EraseLine
This is a Text Editing resource.
Values (Basic or Full): Delete+Meta, BackSpace+Meta

OpenWindows.KeyboardCommand.EraseLineBackward
This mapping is always loaded. This is a Text Editing resource.
Values: u+Ctrl

OpenWindows.KeyboardCommand.EraseLineEnd
This mapping is always loaded. This is a Text Editing resource.
Values: U+Ctrl
**OpenWindows.KeyboardCommand.EraseWordBackward**
This mapping is always loaded. This is a Text Editing resource.
Values: w+Ctrl

**OpenWindows.KeyboardCommand.EraseWordForward**
This mapping is always loaded. This is a Text Editing resource.
Values: W+Ctrl

**OpenWindows.KeyboardCommand.FindBackward**
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: F+Meta, L9+Shift

**OpenWindows.KeyboardCommand.FindForward**
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: f+Meta, L9

**OpenWindows.KeyboardCommand.GoLineForward**
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: apostrophe+Ctrl, R11

**OpenWindows.KeyboardCommand.GoPageBackward**
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: R9

**OpenWindows.KeyboardCommand.GoPageForward**
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: R15

**OpenWindows.KeyboardCommand.GoWordForward**
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: slash+Ctrl, less+Ctrl

**OpenWindows.KeyboardCommand.Help**
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: Help

**OpenWindows.KeyboardCommand.HorizontalScrollbarMenu**
This is a Full “Miscellaneous Navigation” command.
Values: h+Alt

**OpenWindows.KeyboardCommand.IncludeFile**
This mapping is always loaded. This is a Text Editing resource.
Values: i+Meta
OpenWindows.KeyboardCommand.InputFocusHelp
This mapping is loaded if KeyboardCommand is Full. This is a keyboard Core Functions resource.
Values: question+Ctrl

OpenWindows.KeyboardCommand.Insert
This mapping is always loaded. This is a Text Editing resource.
Values: Insert

This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: Down+Ctrl

OpenWindows.KeyboardCommand.JumpLeft
This mapping is always loaded. This is a keyboard Local Navigation command.
Default Values: comma+Ctrl, greater+Ctrl
Values (Basic or Full): Left+Ctrl

OpenWindows.KeyboardCommand.JumpMouseToInputFocus
This mapping is loaded if KeyboardCommand is Full. This is a keyboard Core Functions resource.
Values: j+Alt

OpenWindows.KeyboardCommand.JumpRight
This mapping is always loaded. This is a keyboard Local Navigation command.
Default Values: period+Ctrl
Values (Basic or Full): Right+Ctrl

This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: Up+Ctrl

OpenWindows.KeyboardCommand.Left
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: b+Ctrl, F+Ctrl, Left, R10, Left+Shift
Values (Basic or Full): Left

OpenWindows.KeyboardCommand.LineEnd
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: e+Ctrl, A+Ctrl
OpenWindows.KeyboardCommand.LineStart
This mapping is always loaded. This is a keyboard Local Navigation command.
Values: a+Ctrl, E+Ctrl

OpenWindows.KeyboardCommand.Load
This mapping is always loaded. This is a Text Editing resource.
Values: l+Meta

OpenWindows.KeyboardCommand.MatchDelimiter
This mapping is always loaded. This is a Text Editing resource.
Values: d+Meta

OpenWindows.KeyboardCommand.Menu
This mapping is loaded if KeyboardCommand is Full. This is a keyboard Core Functions resource.
Values: space+Alt

This mapping is always loaded. This is a keyboard Core Functions resource.
Values: Help+Shift

This mapping is always loaded. This is a keyboard Core Functions resource.
Values: Help+Shift+Ctrl

OpenWindows.KeyboardCommand.NextElement
This mapping is loaded if KeyboardCommand is Full. This is a Global Navigation command.
Values: Tab+Ctrl

OpenWindows.KeyboardCommand.NextPane
This mapping is loaded if KeyboardCommand is Full. This is a Global Navigation command.
Values: a+Alt

OpenWindows.KeyboardCommand.PaneBackground
This mapping is loaded if KeyboardCommand is Full. This is a Miscellaneous Navigation command.
Values: b+Alt

OpenWindows.KeyboardCommand.PaneDown
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: R15
OpenWindows.KeyboardCommand.PaneLeft
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: R9+Ctrl

OpenWindows.KeyboardCommand.PaneRight
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: R15+Ctrl

OpenWindows.KeyboardCommand.PaneUp
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: R9

OpenWindows.KeyboardCommand.PanelEnd
This mapping is loaded if KeyboardCommand is Full. This is a Miscellaneous Navigation command.
Values: bracketright+Ctrl

OpenWindows.KeyboardCommand.PanelStart
This mapping is loaded if KeyboardCommand is Full. This is a Miscellaneous Navigation command.
Values: bracketleft+Ctrl

OpenWindows.KeyboardCommand.Paste
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: v+Meta, L8

OpenWindows.KeyboardCommand.PreviousElement
This mapping is loaded if KeyboardCommand is Full. This is a Global Navigation command.
Values: Tab+Shift+Ctrl

OpenWindows.KeyboardCommand.PreviousPane
This mapping is loaded if KeyboardCommand is Full. This is a Global Navigation command.
Values: A+Alt

OpenWindows.KeyboardCommand.Props
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: L3
OpenWindows.KeyboardCommand.QuoteNextKey
This mapping is loaded if KeyboardCommand is Full. This is a keyboard Core Functions resource.
Values: q+Alt

OpenWindows.KeyboardCommand.ResumeMouseless
This mapping is loaded if KeyboardCommand is Full. This is a keyboard Core Functions resource.
Values: Z+Alt

OpenWindows.KeyboardCommand.Right
This mapping is always loaded. This is a keyboard Local Navigation command.
Default Values: f+Ctrl, B+Ctrl, Right, R12, Right+Shift
Values (Basic or Full): Right

OpenWindows.KeyboardCommand.RowEnd
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: End, R13

OpenWindows.KeyboardCommand.RowStart
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Local Navigation command.
Values: Home, R7

OpenWindows.KeyboardCommand.ScrollDataEnd
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: End+Alt+Ctrl, R13+Alt+Ctrl

OpenWindows.KeyboardCommand.ScrollDataStart
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Home+Alt+Ctrl, R7+Alt+Ctrl

OpenWindows.KeyboardCommand.ScrollDown
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Down+Alt

OpenWindows.KeyboardCommand.ScrollJumpDown
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Down+Alt+Ctrl
**OpenWindows.KeyboardCommand.ScrollJumpLeft**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: Left+Alt+Ctrl

**OpenWindows.KeyboardCommand.ScrollJumpRight**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: Right+Alt+Ctrl

**OpenWindows.KeyboardCommand.ScrollJumpUp**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: Up+Alt+Ctrl

**OpenWindows.KeyboardCommand.ScrollLeft**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: Left+Alt

**OpenWindows.KeyboardCommand.ScrollPaneDown**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: R15+Alt

**OpenWindows.KeyboardCommand.ScrollPaneLeft**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: R9+Alt+Ctrl

**OpenWindows.KeyboardCommand.ScrollPaneRight**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: R15+Alt+Ctrl

**OpenWindows.KeyboardCommand.ScrollPaneUp**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: R9+Alt

**OpenWindows.KeyboardCommand.ScrollRight**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.

Values: Right+Alt
OpenWindows.KeyboardCommand.ScrollRowEnd
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: End+Alt, R13+Alt

OpenWindows.KeyboardCommand.ScrollRowStart
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Home+Alt, R7+Alt

OpenWindows.KeyboardCommand.ScrollUp
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Up+Alt

OpenWindows.KeyboardCommand.SelectAll
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: End+Shift+Meta

OpenWindows.KeyboardCommand.SelectDataEnd
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: End+Shift+Ctrl, R13+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectDataStart
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Home+Shift+Ctrl, R7+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectDown
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Down+Shift

OpenWindows.KeyboardCommand.SelectFieldBackward
This mapping is always loaded. This is a Text Editing resource.
Values: Tab+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectFieldForward
This mapping is always loaded. This is a Text Editing resource.
Values: Tab+Ctrl

OpenWindows.KeyboardCommand.SelectJumpDown
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Down+Shift+Ctrl
OpenWindows.KeyboardCommand.SelectJumpLeft
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Left+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectJumpRight
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Right+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectJumpUp
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Up+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectLeft
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Left+Shift

OpenWindows.KeyboardCommand.SelectNextField
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Tab+Meta

OpenWindows.KeyboardCommand.SelectPaneDown
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: R15+Shift

OpenWindows.KeyboardCommand.SelectPaneLeft
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: R9+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectPaneRight
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: R15+Shift+Ctrl

OpenWindows.KeyboardCommand.SelectPaneUp
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: R9+Shift
OpenWindows.KeyboardCommand.SelectPreviousField
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Tab+Shift+Meta

OpenWindows.KeyboardCommand.SelectRight
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Right+Shift

OpenWindows.KeyboardCommand.SelectRowEnd
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: End+Shift, R13+Shift

OpenWindows.KeyboardCommand.SelectRowStart
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Home+Shift, R7+Shift

OpenWindows.KeyboardCommand.SelectUp
This mapping is loaded if KeyboardCommand is Basic or Full. This is a Text Editing resource.
Values: Up+Shift

OpenWindows.KeyboardCommand.Stop
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: L1

OpenWindows.KeyboardCommand.Store
This mapping is always loaded.
Values: s+Meta

OpenWindows.KeyboardCommand.SuspendMouseless
This mapping is loaded if KeyboardCommand is Full. This is a keyboard Core Functions resource.
Values: z+Alt

OpenWindows.KeyboardCommand.TextHelp
This mapping is always loaded. This is a keyboard Core Functions resource.
Values: Help+Ctrl
**OpenWindows.KeyboardCommand.Translate**
This mapping is loaded if KeyboardCommand is Basic or Full. This is a keyboard Core Functions resource.

Values: R2

**OpenWindows.KeyboardCommand.Undo**
This mapping is always loaded. This is a keyboard Core Functions resource.

Values: u+Meta, L4

**OpenWindows.KeyboardCommand.Up**
This mapping is always loaded. This is a keyboard Local Navigation command.

Default Values: p+Ctrl, N+Ctrl, Up, R8, Up+Shift

Values (Basic or Full): Up

**OpenWindows.KeyboardCommand.VerticalScrollbarMenu**
This mapping is loaded if KeyboardCommand is Full. This is a Miscellaneous Navigation command.

Values: v+Alt

**OpenWindows.KeyboardCommands**
All of the OpenWindows.KeyboardCommand resource mappings may be modified by users, or by specifying one of three values for OpenWindows.KeyboardCommands. See the description for the Mouseless model in Chapter 6, Handling Input, in the XView Programming Manual.

Values: Sunview1, Basic, or Full.

**OpenWindows.MonospaceFont**
Compatibility is maintained with the older resources Font.Name and Window.Scale. This sets the font used by cmdtool, shelltool and any text subwindow or text editor that uses a fixed width font. If -font is specified on the command line, then it will override the font set by OpenWindows.MonospaceFont. To find out what fonts are available, use the xlsfonts command. This is a string. The default is Null.

Values: font name

**OpenWindows.MouseChordMenu**
Turns on the mouse chording mechanism. Mouse chording allows XView to work with two button mice. Holding the SELECT and the ADJUST buttons together will act as MENU button.

Values: Boolean (False)

**OpenWindows.MouseChordTimeout**
Mouse chording time-out value.

Values: Integer in microseconds (100)
**OpenWindows.multiClickTimeout (Props)**

Specifies a “click”, which is button-down, button-up pair.

Values: N (4)

Where N is an integer greater than 2. Set the number of tenths of a second between clicks for a multi-click.

**OpenWindows.popupJumpCursor (Props)**

When False, do not warp the mouse to the notice when it appears.

Values: True, False (False)

**OpenWindows.RegularFont**

Sets the font used by panels, menus and notices. If `-font` is specified on the command line, then it will override the font set by `Openwindows.RegularFont`. To find out what fonts are available, use the `xlsfonts` command. The Default is Null.

Values: font name

**OpenWindows.Scale**

Refer to the description for `-scale`.

Values: Small, Medium, Large, Extra-Large

**OpenWindows.scrollbarPlacement (Props)**

When set to Left, put all scrollbars on the left-hand side of the window or object.

Values: Left, Right (Right)

**OpenWindows.SelectDisplaysMenu (Props)**

When True, the SELECT button (usually left mouse) will display the menu as well as the MENU button (usually right mouse).

Values: True, False (False)

**OpenWindows.windowColor (Props)**

Specifies the base color for control areas for a 3-D look. Takes hexadecimal representation. Three other colors used for shading and highlighting are calculated based upon the value of the specified control color. The actual calculated values are done by the OLGX library to provide a consistent color calculation between XView and OLWM. The desktop properties program allows a full range of customization and previews what the chosen 3-D look will look like. Does not apply to monochrome displays.

Values: Any valid X11 color specification (#cccccc — 80% gray)

**OpenWindows.workspaceColor (Props)**

Specifies the color for the root window and the background color for icons that blend into the desktop.

Values: Any valid X11 color specification (#cccccc — 80% gray)

**scrollbar.jumpCursor (Props)**

When False, the scrollbar will not move the mouse pointer when scrolling.

Values: True, False (True)
scrollbar.lineInterval
Specifies the time in milliseconds between repeats of a single line scroll. This indicates how long to pause scrolling when holding down the SELECT button on the scrollbar elevator. Scrollbar sets up a timer routine for repeats.
Values: N (1)
Where N is some integer greater than 0.

scrollbar.pageInterval
Specifies the time in milliseconds between repeats of a single page scroll.
Values: N (100)
Where N is some integer greater than 2.

scrollbar.repeatDelay
Specifies the time in milliseconds when a click becomes a repeated action.
Values: N (100)
Where N is some integer greater than 2.

Selection.Timeout
Selection timeout value. This value indicates the number of seconds that a requestor or a selection owner waits for a response.
Values: Integer (3)

term.alternateTtyswrc
This is only used if a .ttyswrc file is not found in $HOME/.ttyswrc and term.useAlternateTtyswrc is True.
Values: filename ($XVIEWHOME/lib/.ttyswrc)
Where filename specifies a complete filename and absolute path of an alternate .ttyswrc file.

term.boldStyle
Specifies the text bolding style for a terminal-based window.
Values: None, Offset_X, Offset_Y, Offset_X_and_Y_and_XY, Offset_X_and_XY, Offset_Y_and_XY, Offset_X_and_Y, Invert (Invert)

term.enableEdit
When False, do not keep an edit log of what has been typed into the term window. This is set to False automatically when switching from a scrollable term to one that is not scrollable.
Values: True, False (True)

term.inverseStyle
Specifies the text inverting style for a terminal-based window.
Values: Enable, Disable, Same_as_bold (Enable)
term.underlineStyle
Specifies the text underlining style for a terminal-based window.
Values: Enable, Disable, Same_as_bold (Enable)

term.useAlternateTtyswrc
When True, and a $HOME/.ttyswrc is not found, look for an alternate .ttyswrc file.
When False, do not look for an alternate file if one is not found in the home directory, $HOME/.ttyswrc.
Values: True, False (True)

text.againLimit
Number of operations the “again history” remembers for a textsw.
Values: N (1)
Where N is an integer between 0 and 500.

text.autoIndent
When True, begin the next line at the same indentation as the previous line as typing in text.
Values: True, False (False)

text.autoScrollBy
Specifies the number of lines to scroll when type-in moves insertion point below the view.
Values: N (1)
Where N is an integer between 0 and 100.

text.blinkCaret
Values: True, False (True)

text.confirmOverwrite
When False, do not give user confirmation if a save will overwrite an existing file.
Values: True, False (False)

Text.DeleteReplacesClipboard
This resource controls whether text that has been selected and then deleted by the delete key or replaced by any other keystroke will be copied to the clipboard. If the value is True, then the selected text will be copied to the clipboard. If the value is False, then the text selected will not be copied to the clipboard. The default value is False, do not replace the contents of the clipboard.

This resource also applies to the text selected for the filter function. If the resource is True, then the text selected for a filter function will replace the clipboard when the filter successfully finishes. If the resource is False, then the text selected does not replace the clipboard. The default value is False, do not replace the contents of the clipboard.
(in XView V3.0.1, the value for this resource was effectively True, the text selected always replaced the clipboard.)
Values: True, False (False)
**text.delimiterChars**

This resource allows the user to select the delimiter characters that are used when doing word level selections in the XView package. It was added because of the needs of the international marketplace, and it allows the user to define the local delimiters for the character set that is being used with the current keyboard and Sun workstation.

The selection of delimiters will be automatically available to the user once the SunOS 4.1 becomes the default operating system environment, however this resource is used as a bridge during that period.

Note that the octal characters can be scrambled by Xrm during a rewrite of the value of text.delimiterChars. Xrm interprets the text.delimiterChar string when it is loaded. Specifically it will decode the backslashed portions of the string and convert them to octal representations. When this is passed to the client application, the logic will function correctly. However, this misbehavior of Xrm causes the string to be stored incorrectly if the user saves the string. The specific problem(s) that occur are the stripping of the backslash characters and the expansion of the tab character (11).

To correct this problem, one can put the text.delimiterChars entry into an .Xdefaults file that will not be overwritten when saving the workspace properties (for example, a system-wide defaults file). Or a copy of the text.delimiterChars entry can be inserted after .Xdefaults file saves.

Values: string

The default follows:

```
\t,.,;?!'"*+-=/<>\|"#$%^&
```

**text.displayControlChars**

When False, use an up arrow plus a letter to display the control character instead of the character that is available for the current font.

Values: True, False (True)

**text.enableScrollbar**

When False, do not put a scrollbar on the text window.

Values: True, False (True)

**text.extrasMenuFilename**

The file specified by this resource is used for the text package’s Extras menu. This resource can be specified with or without .<locale> appended to the resource name. text.extrasMenuFilename.<locale> will take precedent over text.extrasMenuFilename, where <locale> is the current locale value of the display lang category (XV_LC_DISPLAY/LC_MESSAGES). A command executed in the extras menu is applied to the contents of the current selection in the textsw window. The results are inserted at the current insertion point.

Values: Filename (/usr/lib/.text_extras_menu) where filename is an absolute pathname of a file.
Default: This value may also be set via environment variables EXTRASMENU.<locale> or EXTRASMENU. If neither resource or environment variable exist, the text package will try to locate an Extras menu in one of the following:

$(HOME)/.text_extras_menu.<locale>
$(OPENWINHOME)/lib/locale/<locale>/xview/.text_extras_menu
$(OPENWINHOME)/lib/locale/C/xview/.text_extras_menu
/usr/lib/.text_extras_menu.

**Text.FilterReplacesClipboard.**

If the value is True, then the text selected for a filter command (e.g. the Extras->Format menu item from the textsw menu) will replace the contents of the clipboard after the filter has successfully completed. If the value is False, then the text selected will not replace the contents of the clipboard. The default value is False, do not replace the contents of the clipboard.

Values: True, False (False)

**text.insertMakesCaretVisible**

Controls whether insertion causes repositioning to make inserted text visible.

Values: If_auto_scroll (Always)

**text.lineBreak**

Determines how the textsw treats file lines when they are too big to fit on one display line.

Values: Clip, Wrap_char, Wrap_word (Wrap_word)

**text.lineSpacing**

This resource takes an integer value which is the percentage of the maximum height of a character in the textsw window font to use as interline spacing. For example, the maximum character height in the textsw window font is 10. Setting Text.LineSpacing to 15 will cause 2 pixels (1.5 pixels rounded up) of white space to appear between each line in the textsw.

Values: integer (0)

**text.margin.bottom**

Specifies the minimum number of lines to maintain between insertion point and bottom of view. A value of −1 turns auto scrolling off.

Values: N (0)

Where N is an integer between −1 and 50.

**text.margin.left**

Specifies the margin in pixels that the text should maintain between the left-hand border of the window and the first character on each line.

Values: N (8)

Where N is an integer between 0 and 2000.
**text.margin.right**

Specifies the margin in pixels that the text should maintain between the right-hand border of the window and the last character on each line.

Values:

N (0)

Where N is an integer between 0 and 2000.

**text.margin.top**

Specifies the minimum number of lines to maintain between the start of the selection and the top of the view. A value of –1 means defeat normal actions.

Values:

N (2)

Where N is an integer between –1 and 50.

**text.maxDocumentSize**

Once this limit is exceeded, the text package will send a notice to the user to tell them that no more insertions are possible. If the file being edited is saved to a file, or it is a disk file being edited, then the limit does not apply.

Values:

N (20000)

Where N specifies the bytes used in memory before a text file is saved to a file on disk.

**text.retained**

If True, retain text windows with server backing store.

Values:

True, False (False)

**text.storeChangesFile**

When False, do not change the name of the current file being edited to the name of the file that is stored. The name of the current file is reflected in the titlebar of the textedit frame.

Values:

True, False (True)

**text.tabWidth**

Specifies the width in characters of the tab character.

Values:

N (8)

Where N is an integer between 0 and 50.

**text.undoLimit**

Specifies how many operations to save in the undo history log. These operations will be undone when you press the “Undo” key in the text window.

Values:

N (50 maximum of 500)

Where N is an integer between 0 and 500.

**tty.eightBitOutput**

This resource controls whether characters modified by the meta modifier are encoded as eight-bit characters when passed to the ttysw’s pty, or are delivered as seven-bit characters.

Values:

Boolean

Default is True.
tty.yieldModifiers
Any semantic meaning, mouseless command or keyboard accelerator, that would normally be associated with the listed modifiers when the keyboard focus is in a ttysw or a termsw would be removed. The default is to not remove any semantic meaning from any modifiers.

Values: List of modifier keys
         Alt and Meta are the only two keys currently supported.

window.synchronous, +sync -sync
Useful when debugging or tracking down a problem since the error codes emitted from Xlib will correspond to the immediate request made. Running in a synchronous mode will cause the application to run significantly slower.

Values: True, False (False)

Window.Depth
Specifies the window’s depth. Use an integer.

Window.visual
Resource specifies the visual used for the window.

xview.icccmcompliant
When False, XView will set window manager hints in a way that was used before the ICCCM was adopted. Useful for window managers that are released before X11R4. Not needed with the Open Look Window Manager provided with Open Windows.

Values: True, False (True)
This page intentionally left blank

to preserve original page counts.
A

Selection Compatibility Attributes

This appendix lists all the selection compatibility attributes in alphabetical order. Appendix B, Selection Compatibility Procedures and Macros, lists the selection compatibility procedures. XView Version 3 supports a newer selection mechanism that implements selections using the SELECTION package. The selection mechanism that was available in older versions of XView is supported in XView Version 3; its attributes are shown in this appendix and the mechanism is described in Appendix A, The Selection Service, in the XView Programming Manual. All of these compatibility attributes use the SELN_-prefix. These attributes do not work with the XView SELECTION package.

Each selection attribute’s description is in the format below.

INTRODUCTION
This field provides a brief description of the attribute.

Argument: This field shows the first programmer-supplied value associated with the attribute. If an attribute has multiple values, then the type of each value is shown in multiple Argument fields.

Default: The default field shows the default value for the attribute.

Procs: The procedures field shows the procedures that are valid for the attribute. Note: These attributes are not valid for Selection objects.

SELN_REQ_BYTESIZE
Specifies the number of bytes in the selection.

Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query() selection_commit_pending_delete()

SELN_REQ_COMMIT_PENDING_DELETE
Instructs the replier to delete any secondary selection made in pending delete mode.

Argument: None
Default: None
**SELN_REQ_CONTENTS_ASCII**
Specifies a NULL-terminated list of 4-byte words containing the selection’s ASCII contents. If the last word of the contents is not full (including NULL terminator for the string), it is NULL-padded.

Argument: char *
Default: None
Procs: selection_ask() selection_init_request() selection_query()

**SELN_REQ_CONTENTS_PIECES**
Specifies that the value is a NULL-terminated list of 4-byte words containing the selection’s contents described in the textsw’s piece-table format.

Argument: char *
Default: None
Procs: selection_ask() selection_init_request() selection_query()

**SELN_REQ_DELETE**
Instructs the holder of the selection to delete the contents of the selection from its window (used only by text subwindows).

Argument: void
Default: None
Procs: selection_ask() selection_init_request() selection_query()

**SELN_REQ_END_REQUEST**
Returns an error for failed or unrecognized requests.

Argument: void
Default: None
Procs: selection_ask() selection_init_request() selection_query()

**SELN_REQ_FAKE_LEVEL**
Gives a level to which the selection should be expanded before processing the remainder of this request. The original level should be maintained on the display, however, and restored as the true level on completion of the request.

Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()

**SELN_REQ_FILE_NAME**
Specifies a NULL-terminated list of 4-byte words. Contains the name of the file which holds the shelf selection.

Argument: char *
Default: None
Procs: selection_ask() selection_init_request() selection_query()

**SELN_REQ_FIRST**
Gives the number of bytes that precede the first byte of the selection.

Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()
SELN_REQ_FIRST_UNIT
Gives the number of units of the selection’s current level (line, paragraph, etc.) which precede the first unit of the selection.
Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()

SELN_REQ_LAST
Gives the byte index of the last byte of the selection.
Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()

SELN_REQ_LAST_UNIT
Gives the unit index of the last unit of the selection at its current level.
Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()

SELN_REQ_LEVEL
Gives the current level of the selection.
Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()

SELN_REQ_RESTORE
Instructs the replier to restore the selection referred to in this request, if it has maintained sufficient information to do so.
Argument: No value
Default: None
Procs: selection_ask() selection_init_request() selection_query()

SELN_REQ_SET_LEVEL
Gives a level to which the selection should be set. This request should affect the true level.
Argument: int
Default: None
Procs: selection_ask() selection_init_request() selection_query()

SELN_REQ_YIELD
Requests the holder of the selection to yield it. SELN_SUCCESS, SELN_DIDNT_HAVE, and SELN_WRONG_RANK are legitimate responses. The latter comes from a holder asked to yield the primary selection when it knows a function key is down.
Argument: Seln_result
Default: None
Procs: selection_ask() selection_init_request() selection_query()
This page intentionally left blank

to preserve original page counts.
This section lists the XView Selection procedures and macros in alphabetical order. If you are creating new applications with XView Version 3, you should not use these functions (refer to Chapter 18, Selections, in the XView Programming Manual for further information).

**Selection Compatibility Procedures and Macros**

### selection_acquire()

Acquires the selection of a specified rank. This is typically used internally by XView packages. It is not used to inquire about the current selection.

```c
Seln_rank
selection_acquire(server, client, asked)
Xv_Server server;
Seln_client client;
Seln_rank asked;
```

*client* is the opaque handle returned from *selection_create()*. The client uses this call to become the new holder of the selection of rank *asked*. *asked* should be one of SELN_CARET, SELN_PRIMARY, SELN_SECONDARY, SELN_SHELF, or SELN_UNSPECIFIED. If successful, the rank actually acquired is returned.

If *asked* is SELN_UNSPECIFIED, the client indicates it wants whichever of the primary or secondary selections is appropriate given the current state of the function keys; the one acquired can be determined from the return value.

### selection_ask()

`selection_ask()` is a simplified form of `selection_request()` that looks and acts very much like `seln_query()`. The only difference is that it does not use a callback *proc* and so cannot handle replies that require more than a single buffer (e.g., long text selections). If it receives a reply consisting of more than one buffer, it returns the first buffer and discards the rest. The return value is a pointer to a static buffer; in case of error, this will be a valid pointer to a NULL buffer:

```c
buffer->status = SELN_FAILED
```

The call looks like this:

```c
Seln_request *
selection_ask(server, holder, attributes, ..., NULL)
Xv_Server server;
```
Seln_holder *holder;
Attr_union attributes;

**selection_clear_functions()**

The server is told to forget about any function keys it thinks are down, resetting its state to all-up. If it knows of a current secondary selection, the server will tell its holder to yield.

```c
void
selection_clear_functions()
```

**selection_create()**

The server is initialized for this client. Client_data is a 32-bit opaque client value which the server will pass back in callback procedures, as described above. The first two arguments are addresses of client procedures which will be called from the selection functions when client processing is required. These occasions occur when the server sees a function-key transition which may interest this client and when another process wishes to make a request concerning the selection this client holds.

```c
Seln_client
selection_create(
    server, function_proc, request_proc, client_data)
    Xv_Server, server;
    void (*function_proc) ();
    Seln_result (*request_proc) ();
    Xv_opaque client_data;
```

**selection_destroy()**

A client created by selection_create is destroyed—any selection it may hold is released, and various pieces of data associated with the selection mechanism are freed. If this is the last client in this process using the Selection Service, the RPC socket is closed and its notification removed.

```c
void
selection_destroy(server, client)
    Xv_Server server;
    Seln_client client;
```

**selection_done()**

Client indicates it is no longer the holder of the selection of the indicated rank. The only cause of failure is absence of the server. It is not necessary for a client to call this procedure when it has been asked by the server to yield a selection.

```c
Seln_result
selection_done(server, client, rank)
    Xv_Server server;
    Seln_client client;
    Seln_rank rank;
```
**selection_figure_response()**

Procedure to determine the correct response according to the standard user interface when `seln_inform()` returns *buffer or the client’s function_procs called with it. The addressee_rank field in Seln_function_buffer will be modified to indicate the selection which should be affected by this client; holder will be set to point to the element of *buffer which should be contacted in the ensuing action, and the return value indicates what that action should be. Possible return values are SELN_DELETE, SELN_FIND, SELN_IGNORE, SELN_REQUEST, and SELN_SHELVE.

```c
Seln_response
selection_figure_response(server, buffer, holder)
Xv_Server server;
Seln_function_buffer *buffer;
Seln_holder *holder;
```

**selection_hold_file()**

The server is requested to act as the holder of the specified rank, whose ASCII contents have been written to the file indicated by path. This allows a selection to persist longer than the application which made it can maintain it. Most commonly, this will be done by a process which holds the shelf when it is about to terminate.

```c
Seln_result
selection_hold_file(server, rank, path)
Xv_Server server;
Seln_rank rank;
char *path;
```

**selection_inform()**

Low-level, policy-independent procedure for informing the server that a function key has changed state. Most clients will prefer to use the higher-level procedure `seln_report_event`, which handles much of the standard interpretation required.

```c
Seln_function_buffer
selection_inform(server, client, which, down)
Xv_Server server;
Seln_client client;
Seln_function which;
int down;
```

**selection_init_request()**

Procedure used to initialize a buffer before calling `selection_request`. (It is also called internally by `selection_ask` and `seln_query`.) It takes a pointer to a request buffer, a pointer to a struct referring to the selection holder to which the request is to be addressed, and a list of attributes which constitute the request to be sent. The attributes are copied into buffer->data, and the corresponding size is stored into buffer->buf_size. Both elements of requester_data are zeroed; if the caller wants to handle long requests, consumer-proc and context pointers must be entered in these elements after `selection_init_request` returns.

```c
void
selection_init_request(
    server, buffer, holder, attributes, ..., NULL)
```
selection_inquire()

Returns a Seln_holder structure containing information which enables the holder of the indicated selection to be contacted. If the rank argument is SELN_UNSPECIFIED, the server will return access information for either the primary or the secondary selection holder, as warranted by the state of the function keys it knows about. The rank element in the returned struct will indicate which is being returned.

This procedure may be called without selection_create() having been called first.

Seln_holder
selection_inquire(server, rank)
  Xv_Server      server;
  Seln_rank      rank;

selection_inquire_all()

Returns a Seln_holders_all struct from the Selection Service; it consists of a Seln_holder struct for each of the four ranks.

Seln_holders_all
selection_inquire_all()

selection_query()

Transmits a request to the selection holder indicated by the holder argument. consume and context are used to interpret the response and are described below. The remainder of the arguments to selection_query constitute an attribute-value list which is the request. (The last argument should be a 0 to terminate the list.) The procedure pointed to by reader will be called repeatedly with a pointer to each buffer of the reply. The value of the context argument will be available in buffer->requester_data.context for each buffer. This item is not used by the selection library; it is provided for the convenience of the client. When the reply has been completely processed (or when the consume proc returns something other than SELN_SUCCESS), selection_query returns.

Selection_result
selection_query(server,holder,reader,context,attributes,...,NULL)
  Xv_Server      server;
  Seln_holder    *holder;
  Seln_result    (*reader)();
  char           *context;
  A-V list       attributes;
selection_report_event()

High-level procedure for informing the server of a function key transition which may affect the selection. It incorporates some of the policy of the standard user interface and provides a more convenient interface to selection_inform.

Seln_client_node is the client handle returned from selection_create; it may be 0 if the client guarantees it will not need to respond to the function transition.

Event is a pointer to the struct inputevent which reports the transition seln_report_event. selection_report_event generates a corresponding call to seln_inform and, if the returned struct is not NULL, passes it to the client’s function_proc callback procedure.

```c
void
selection_report_event(server, client, event)
    Xv_Server server;
    Seln_client_node *client;
    Event *event;
```

selection_request()

Low-level, policy-independent mechanism for retrieving information about a selection from the server. Most clients will access it only indirectly, through selection_ask or selection_query.

selection_request takes a pointer to a holder (as returned by seln_inquire) and a request constructed in *buffer. The request is transmitted to the indicated selection holder, and the buffer rewritten with its response. Failures in the RPC mechanism will cause a SELN_FAILED return; if the process of the addressed holder is no longer active, the return value will be SELN_NON_EXIST. Clients which call selection_request directly will find it most convenient to initialize the buffer by a call to selection_init_request.

Request attributes which are not recognized by the selection holder will be returned as the value of the attribute SELN_UNRECOGNIZED. Responses should be provided in the order requests were encountered.

```c
Seln_result
selection_request(server, holder, buffer)
    Xv_Server server
    Seln_holder *holder;
    Seln_request *buffer;
```

selection_yield_all()

Procedure that queries the holders of all selections and, for each which is held by a client in the calling process, sends a yield request to that client and a Done to the server. It should be called by applications which are about to exit or to undertake lengthy computations during which they will be unable to respond to requests concerning selections they hold.

```c
void
selection_yield_all()
```
This page intentionally left blank
to preserve original page counts.
This page intentionally left blank

to preserve original page counts.
This appendix lists all the textsw ACTION_ attributes, in alphabetical order. When an application defines a text subwindow notify procedure and sets TEXTSW_NOTIFY_PROC, the notify procedure may handle any of the attributes shown in this appendix. These attributes are not valid for xv_create(), xv_get(), or xv_set(). They are provided for application (client) supplied notify procedures. For more information, see the enum Textsw_action in textsw.h. The first entry describes the format used for the attributes in this appendix.

**INTRODUCTION**
This field provides a brief description of the attribute.

**Argument:** This field shows the first programmer-supplied value associated with the attribute. If an attribute has multiple values, then the type of each value is shown in multiple Argument fields.

**Object:** This shows the object that the procedure is valid for.

**TEXTSW_ACTION_CAPS_LOCK**
The user pressed the Caps Lock key to change the setting of the Caps Lock (it is initially 0, meaning off).

**Argument:** Boolean

**Objects:** textsw

**TEXTSW_ACTION_CHANGED_DIRECTORY**
The current working directory for the process has been changed to the directory named by the provided string value.

**Argument:** char *

**Objects:** textsw

**TEXTSW_ACTION_EDITED_FILE**
The file named by the provided string value has been edited. Appears once per session of edits (see below).

**Argument:** char *

**Objects:** textsw

**TEXTSW_ACTION_EDITED_MEMORY**
Monitors whether an empty text subwindow has been edited.

**Argument:** No value

**Objects:** textsw
TEXTSW_ACTION_FILE_IS_READONLY
The file named by the provided string value does not have write permission.
Argument: char *
Objects: textsw

TEXTSW_ACTION_LOADED_FILE
The text subwindow is being used to view the file named by the provided string value.
Argument: char *
Objects: textsw

TEXTSW_ACTION_TOOL_CLOSE
The frame containing the text subwindow should become iconic.
Argument: No value
Objects: textsw

TEXTSW_ACTION_TOOL_DESTROY
The tool containing the text subwindow should exit, without checking for a veto from other subwindows. The value is the user action that caused the destroy.
Argument: Event *
Objects: textsw

TEXTSW_ACTION_TOOL_QUIT
The tool containing the text subwindow should exit normally. The value is the user action that caused the exit.
Argument: Event *
Objects: textsw

TEXTSW_ACTION_TOOL_MGR
The tool containing the text subwindow should do the window manager operation associated with the provided event value.
Argument: Event *
Objects: textsw

TEXTSW_ACTION_USING_MEMORY
The text subwindow is being used to edit a string stored in primary memory, not a file.
Argument: No value
Objects: textsw
abbreviated choice item, 17
attr_create_list() procedure, 175
attribute, ATTR_LIST, 32
    CANVAS_AUTO_CLEAR, 32
    CANVAS_AUTO_EXPAND, 32
    CANVAS_AUTO_SHRINK, 32
    CANVAS_CMS_REPAINT, 32
    CANVAS_FIXED_IMAGE, 32
    CANVAS_HEIGHT, 33
    CANVAS_MIN_PAINT_HEIGHT, 33
    CANVAS_MIN_PAINT_WIDTH, 33
    CANVAS_NO_CLIPPING, 33
    CANVAS_NTH_PAINT_WINDOW, 33
    CANVAS_PAINT_CANVAS_WINDOW, 34
    CANVAS_PAINT_VIEW_WINDOW, 34
    CANVAS_PAINTWINDOW_ATTRS, 34
    CANVAS_REPAINT_PROC, 34
    CANVAS_RESIZE_PROC, 35
    CANVAS RETAINED, 35
    CANVAS_VIEWABLE_RECT, 36
    CANVAS_VIEW_CANVAS_WINDOW, 36
    CANVAS_VIEW_PAINT_WINDOW, 36
    CANVAS_WIDTH, 36
    CANVAS_X_PAINT_WINDOW, 36
    CMS_BACKGROUND_PIXEL, 37
    CMS_COLOR_COUNT, 37
    CMS COLORS, 37
    CMS_CONTROL_CMS, 37
    CMS_FOREGROUND_PIXEL, 38
    CMS_INDEX, 38
    CMS_INDEX_TABLE, 38
    CMS_NAME, 38
    CMS_NAMED COLORS, 38
    CMS_PIXEL, 38
    CMS_SCREEN, 39
    CMS_SIZE, 39
    CMS_TYPE, 39
    CMS_X COLORS, 39
    CURSOR_BACKGROUND_COLOR, 39
    CURSOR_DRAG_STATE, 39
    CURSOR_DRAG_TYPE, 40
    CURSOR_FOREGROUND_COLOR, 40
    CURSOR_IMAGE, 40
    CURSOR_MASK_CHAR, 40
    CURSOR_OP, 40
    CURSOR_SRC_CHAR, 41
    CURSOR_STRING, 41
    CURSOR_XHOT, 41
    CURSOR_YHOT, 41
    DND_ACCEPT_CURSOR, 41
    DND_ACCEPT_X_CURSOR, 42
    DND_CURSOR, 42
    DND_TIMEOUT_VALUE, 42
    DND TYPE, 42
    DND X_CURSOR, 42
    DROP_SITE_DEFAULT, 43
    DROP_SITE_DELETE_REGION, 43
    DROP_SITE_DELETE_REGION_PTR, 43
    DROP_SITE_EVENT_MASK, 43
    DROP_SITE_REGION, 43
    DROP_SITE_REGION_PTR, 44
    FILE_CHOOSER_ABBREV_VIEW, 44
    FILE_CHOOSER_APP_DIR, 44
    FILE_CHOOSER_AUTO_UPDATE, 45
    FILE_CHOOSER_CD_FUNC, 45
    FILE_CHOOSER_CHILD, 45
    FILE_CHOOSERCOMPARE_FUNC, 46
    FILE_CHOOSER_CUSTOMIZE_OPEN, 46
    FILE_CHOOSER_DIRECTORY, 47
    FILE_CHOOSER_DOC_NAME, 47
    FILE_CHOOSER_EXTEN_FUNC, 47
    FILE_CHOOSER_EXTEN_HEIGHT, 48
    FILE_CHOOSER_FILTER_FUNC, 48
    FILE_CHOOSER_FILTER_MASK, 49
    FILE_CHOOSER_FILTER_STRING, 49
    FILE_CHOOSER_HISTORY_LIST, 49
    FILE_CHOOSER_MATCH_GLYPH, 50
attribute (cont’d)
  FILE_CHOOSER_MATCH_GLYPH_MASK, 50
  FILE_CHOOSER_NO_CONFIRM, 50
  FILE_CHOOSER_NOTIFY_FUNC, 50
  FILE_CHOOSER_SAVE_TO_DIR, 51
  FILE_CHOOSER_SHOW_DOT_FILES, 51
  FILE_CHOOSER_TYPE, 51
  FILE_CHOOSER_UPDATE, 51
  FILE_LIST_ABBREVVIEW, 52
  FILE_LIST_AUTO_UPDATE, 52
  FILE_LIST_CHANGE_DIR_FUNC, 52
  FILE_LIST_COMPARE_FUNC, 53
  FILE_LIST_DIRECTORY, 53
  FILE_LIST_DOTDOT_STRING, 53
  FILE_LIST_FILTER_FUNC, 54
  FILE_LIST_FILTER_MASK, 55
  FILE_LIST_FILTER_STRING, 53
  FILE_LIST_MATCH_GLYPH, 55
  FILE_LIST_MATCH_GLYPH_MASK, 55
  FILE_LIST_ROW_TYPE, 55
  FILE_LIST_SHOW_DIR, 56
  FILE_LIST_SHOW_DOT_FILES, 56
  FILE_LIST_UPDATE, 56
  FILE_LIST_USE_FRAME, 56
  FONT_CHAR_HEIGHT, 57
  FONT_CHAR_WIDTH, 57
  FONT_DEFAULT_CHAR_HEIGHT, 57
  FONT_DEFAULT_CHAR_WIDTH, 57
  FONT_FAMILY, 57
  FONT_INFO, 57
  FONT_NAME, 58
  FONT_PIXFONT, 58
  FONT_RESCALE_OF, 58
  FONT_SCALE, 58
  FONT_SIZE, 59
  FONT_SIZES_FOR_SCALE, 59
  FONT_STRING_DIMS, 60
  FONT_STYLE, 60
  FRAME_ACCELERATOR, 60
  FRAME_BACKGROUND_COLOR, 61
  FRAME_BUSY, 61
  FRAME_CLOSED, 61
  FRAME_CLOSED_RECT, 61
  FRAME_CMD_DEFAULT_PIN_STATE, 61
  FRAME_CMD_PANEL, 62
  FRAME_CMD_PIN_STATE, 62
  FRAME_CMD_PUSH_PIN_IN, 62
  FRAME_DEFAULT_DONE_PROC, 62
  FRAME_DONE_PROC, 62
  FRAME_FOCUS_DIRECTION, 63
  FRAME_FOCUS_WIN, 63
  FRAME_FOREGROUND_COLOR, 63
  FRAME_ICON, 63
  FRAME_INHERIT_COLORS, 63
  FRAME_LABEL, 63
  FRAME_LEFT FOOTER, 63
  FRAME_MAX_SIZE, 64
  FRAME_MENU_ADD, 64
  FRAME_MENU_COUNT, 64
  FRAME_MENU_DELETE, 65
  FRAME_MENUS, 64
  FRAME_MIN_SIZE, 65
  FRAME_NEXT_PANE, 65
  FRAME_NO_CONFIRM, 66
  FRAME_NTH_SUBFRAME, 66
  FRAME_NTH_SUBWINDOW, 66
  FRAME_PREVIOUS_ELEMENT, 66
  FRAME_PREVIOUS_PANE, 66
  FRAME_RIGHT_FOOTER, 66
  FRAME_SHOW_FOOTER, 67
  FRAME_SHOW_HEADER, 67
  FRAME_SHOW_LABEL, 67
  FRAME_SHOW_RESIZE_CORNER, 67
  FRAME_WM_COMMAND_ARGC, 67
  FRAME_WM_COMMAND_ARGC_ARGV, 68
  FRAME_WM_COMMAND_ARGV, 69
  FRAME_WM_COMMAND_STRINGS, 69
  FRAME_X_ACCELERATOR, 70
  FULLSCREEN_ALLOW_EVENTS, 70
  FULLSCREEN.Allow SYNC EVENT, 70
  FULLSCREEN_CURSOR_WINDOW, 70
  FULLSCREEN_GRAB_KEYBOARD, 70
  FULLSCREEN_GRAB_POINTER, 71
  FULLSCREEN_GRAB_SERVER, 71
  FULLSCREEN_INPUT_WINDOW, 71
  FULLSCREEN_OWNER_EVENTS, 71
  FULLSCREEN_PAINT_WINDOW, 72
  FULLSCREEN_POINTER_GRAB_KBD_MODE, 72
  FULLSCREEN_RECT, 72
  FULLSCREEN_SYNC, 72
  HELP_STRING_FILENAME, 72
  HISTORY_ADD_FIXED_ENTRY, 73
  HISTORY_ADD_ROLLING_ENTRY, 73
  HISTORY_DUPLICATE_LABELS, 73
  HISTORY_DUPLICATE_VALUES, 73
  HISTORY_FIXED_COUNT, 73
  HISTORY_INACTIVE, 74
  HISTORY_LABEL, 74
  HISTORY_MENU_HISTORY_LIST, 74

XView Reference Manual
**attribute (cont’d)**

- HISTORY_MENU_OBJECT, 74
- HISTORY_NOTIFY_PROC, 74
- HISTORY_ROLLING_COUNT, 75
- HISTORY_ROLLING_MAXIMUM, 75
- HISTORY_VALUE, 75
- ICON_FONT, 75
- ICON_HEIGHT, 75
- ICON_IMAGE, 76
- ICON_IMAGE_RECT, 76
- ICON_LABEL, 76
- ICON_LABEL_RECT, 76
- ICON_MASK_IMAGE, 76
- ICON_TRANSPARENT, 76
- ICON_TRANSPARENT_LABEL, 77
- ICON_WIDTH, 77
- MENU_ACCELERATOR, 77
- MENU_ACTION_ACCELERATOR, 77
- MENU_ACTION_IMAGE, 78
- MENU_ACTION_ITEM, 78
- MENU_APPEND_ITEM, 78
- MENU_CLASS, 78
- MENU_CLIENT_DATA, 78
- MENU_COL_MAJOR, 79
- MENU_COLOR, 78
- MENU_DEFAULT, 79
- MENU_DEFAULT_ITEM, 79
- MENU_DESCEND_FIRST, 79
- MENU_DONE_PROC, 79
- MENU_FEEDBACK, 80
- MENU_FIRST_EVENT, 80
- MENU_GEN_PIN_WINDOW, 80
- MENU_GEN_PROC, 80
- MENU_GEN_PROC_IMAGE, 80
- MENU_GEN_PROC_ITEM, 81
- MENU_GEN_PULLRIGHT, 81
- MENU_GEN_PULLRIGHT_IMAGE, 81
- MENU_GEN_PULLRIGHT_ITEM, 81
- MENU_IMAGE, 81
- MENU_IMAGE_ITEM, 81
- MENU_IMAGES, 81
- MENU_INACTIVE, 82
- MENU_INSERT, 82
- MENU_INSERT_ITEM, 82
- MENU_ITEM, 82
- MENU_LAST_EVENT, 82
- MENU_NCHARS, 83
- MENU_NITEMS, 83
- MENU_NOTIFY_PROC, 83
- MENU_NOTIFY_STATUS, 83
- MENU_NROWS, 84
- MENU_NTH_ITEM, 84
- MENU_PARENT, 84
- MENU_PIN, 84
- MENU_PIN_PROC, 84
- MENU_PIN_WINDOW, 85
- MENU_PULLRIGHT, 85
- MENU_PULLRIGHT_IMAGE, 85
- MENU_PULLRIGHT_ITEM, 85
- MENU_RELEASE, 85
- MENU_RELEASE_IMAGE, 85
- MENU_REMOVE, 85
- MENU_REPLACE, 86
- MENU_REPLACE_ITEM, 86
- MENU_SELECTED, 86
- MENU_SELECTED_ITEM, 86
- MENU_STRINGS, 86
- MENU_STRING_ITEM, 86
- MENU_STRINGS, 87
- MENU_STRINGS_AND_ACCELERATORS, 87
- MENU_TITLE, 87
- MENU_TITLE_ITEM, 87
- MENU_TYPE, 87
- MENU_VALID_RESULT, 88
- MENU_VALUE, 88
- NOTICE_BLOCK_THREAD, 88
- NOTICE_BUSY_FRAMES, 88
- NOTICE_BUTTON, 88
- NOTICE_BUTTON_NO, 88
- NOTICE_BUTTON_YES, 89
- NOTICE_EVENT_PROC, 89
- NOTICE_FOCUS_XY, 89
- NOTICE_FONT, 90
- NOTICE_LOCK_SCREEN, 90
- NOTICE_MESSAGE_STRING, 90
- NOTICE_MESSAGE_STRINGS, 90
- NOTICE_MESSAGE_STRINGS_ARRAY_PTR, 90
- NOTICE_NO_BEEPING, 90
- NOTICE_STATUS, 91
- NOTICE_TRIGGER, 91
- NOTICE_TRIGGER_EVENT, 91
- OPENWIN_ADJUST_FOR_HORIZONTAL_SCROLLBAR, 91
- OPENWIN_ADJUST_FOR_VERTICAL_SCROLLBAR, 91
- OPENWIN_AUTO_CLEAR, 92
- OPENWIN_HORIZONTAL_SCROLLBAR, 92
- OPENWIN_NO_MARGIN, 92
- OPENWIN_NTH_VIEW, 92
- OPENWIN_NVIEWS, 92
- OPENWIN_SHOW_BORDERS, 93
- OPENWIN_SPLIT, 93
- OPENWIN_SPLIT_DESTROY_PROC, 93
- OPENWIN_SPLIT_DIRECTION, 93
attribute (cont'd)

OPENWIN_SPLIT_INIT_PROC, 93
OPENWIN_SPLIT_POSITION, 94
OPENWIN_SPLIT_VIEW, 94
OPENWIN_SPLIT_VIEW_START, 94
OPENWIN_VERTICAL_SCROLLBAR, 94
OPENWIN_VIEW_ATTRS, 94
PANEL_ACCEPT_KEYSTROKE, 95
PANEL_BACKGROUND_PROC, 95
PANEL_BLINK_CARET, 95
PANEL_BORDER, 95
PANEL_BUSY, 96
PANEL_CARET_ITEM, 96
PANEL_CHOICE_COLOR, 96
PANEL_CHOICE_FONT, 96
PANEL_CHOICE_FONTS, 96
PANEL_CHOICE_IMAGE, 97
PANEL_CHOICE_IMAGES, 97
PANEL_CHOICE_NCOLS, 97
PANEL_CHOICE_NROWS, 97
PANEL_CHOICE_RECT, 97
PANEL_CHOICE_STRING, 97
PANEL_CHOICE_STRINGS, 98
PANEL_CHOICE_X, 98
PANEL_CHOICE_XS, 98
PANEL_CHOICE_Y, 98
PANEL_CHOICE_YS, 98
PANEL_CHOICE_NONE, 98
PANEL_CHOICE_ONE, 98
PANEL_CLIENT_DATA, 98
PANEL_CURRENT_ITEM, 99
PANEL_DEFAULT_ITEM, 99
PANEL_DEFAULT_VALUE, 99
PANEL_DIRECTION, 99
PANEL_DISPLAY_LEVEL, 99
PANEL_DISPLAY_ROWS, 100
PANEL_DROP_BUSY_GLYPH, 100
PANEL_DROP_DND, 100
PANEL_DROP_FULL, 100
PANEL_DROP_GLYPH, 100
PANEL_DROP_HEIGHT, 100
PANEL_DROP_SEL_REQ, 101
PANEL_DROP_SITE_DEFAULT, 101
PANEL_DROP_WIDTH, 101
PANEL_EVENT_PROC, 101
PANEL_EXTRA_PAINT_HEIGHT, 101
PANEL_EXTRA_PAINT_WIDTH, 102
PANEL_FEEDBACK, 102
PANEL_FIRST_ITEM, 102
PANEL_FIRST_PAINT_WINDOW, 102
PANEL_FOCUS_PW, 102
PANEL_GAUGE_WIDTH, 102
PANEL_GINFO, 103
PANEL_INACTIVE, 103
PANEL_ITEM_CLASS, 103
PANEL_ITEM_COLOR, 103
PANEL_ITEM_CREATED, 103
PANEL_ITEM_DEAF, 103
PANEL_ITEM_LABEL_RECT, 104
PANEL_ITEM_MENU, 104
PANEL_ITEM_NTH_WINDOW, 104
PANEL_ITEM_NWINDOWS, 104
PANEL_ITEM_OWNER, 104
PANEL_ITEM_RECT, 105
PANEL_ITEM_VALUE_RECT, 105
PANEL_ITEM_WANTS_ADJUST, 105
PANEL_ITEM_WANTS_ISO, 105
PANEL_ITEM_X, 105
PANEL_ITEM_X_GAP, 105
PANEL_ITEM_X_POSITION, 106
PANEL_ITEM_Y, 106
PANEL_ITEM_Y_GAP, 106
PANEL_ITEM_Y_POSITION, 106
PANEL_JUMP_DELTA, 106
PANEL_LABEL_BOLD, 107
PANEL_LABEL_FONT, 107
PANEL_LABEL_IMAGE, 107
PANEL_LABEL_STRING, 107
PANEL_LABEL_WIDTH, 107
PANEL_LABEL_X, 108
PANEL_LABEL_Y, 108
PANEL_LAYOUT, 108
PANEL_LINE_BREAK_ACTION, 108
PANEL_LIST_CLIENT_DATA, 108
PANEL_LIST_CLIENT_DATAS, 109
PANEL_LIST_DISPLAY_ROWS, 109
PANEL_LIST_DO_DBLCCLICK, 110
PANEL_LIST_EXTENSION_DATA(S), 110
PANEL_LIST_FIRST_SELECTED, 110
PANEL_LIST_FONT, 110
PANEL_LIST_GLYPH, 111
PANEL_LIST_GLYPHS, 111
PANEL_LIST_INACTIVE, 111
PANEL_LIST_INSERT, 111
PANEL_LIST_INSERT_DUPPLICATE, 111
PANEL_LIST_INSERT_GLYPHS, 112
PANEL_LIST_INSERT_STRINGS, 112
PANEL_LIST_MASK_GLYPH, 112
PANEL_LIST_MASK_GLYPHS, 112
attribute (cont’d)

PANEL_LIST_MODE, 112
PANEL_LIST_NEXT_SELECTED, 113
PANEL_LIST_NROWS, 113
PANEL_LIST_ROW_HEIGHT, 113
PANEL_LIST_SCROLLBAR, 114
PANEL_LIST_SELECT, 114
PANEL_LIST_SELECTED, 114
PANEL_LIST_SORT, 114
PANEL_LIST_STRING, 114
PANEL_LIST_STRINGS, 115
PANEL_LIST_TITLE, 115
PANEL_LIST_WIDTH, 115
PANEL_MASK_CHAR, 115
PANEL_MAX_TICK_STRING, 115
PANEL_MAX_VALUE, 116
PANEL_MAX_VALUE_STRING, 116
PANEL_MIN_TICK_STRING, 116
PANEL_MIN_VALUE, 116
PANEL_MIN_VALUE_STRING, 116
PANEL_NCHOICES, 117
PANEL_NEXT_COL, 117
PANEL_NEXT_ITEM, 117
PANEL_NEXT_ROW, 117
PANEL_NO_REDISPLAY_ITEM, 117
PANEL_NOTIFY_LEVEL, 118
PANEL_NOTIFY_PROC, 118
PANEL_NOTIFY_STATUS, 120-121
PANEL_NOTIFY_STRING, 121
PANEL_OPS_VECTOR, 121
PANEL_PRIMARY_FOCUS_ITEM, 121
PANEL_READ_ONLY, 122
PANEL_REPAINT_PROC, 122
PANEL_SHOW_RANGE, 122
PANEL_SHOW_VALUE, 122
PANEL_SLIDER_END_BOXES, 122
PANEL_SLIDER_WIDTH, 123
PANEL_STATUS, 123
PANEL_TEXT_SELECT_LINE, 123
PANEL.Ticks, 123
PANEL_TOGGLE_VALUE, 123
PANEL.VALUE, 123
PANEL.VALUE_DISPLAY_LENGTH, 21, 124
PANEL.VALUE_DISPLAY_WIDTH, 21, 124
PANEL.VALUE_FONT, 124
PANEL.VALUE_STORED_LENGTH, 21, 124
PANEL.VALUE_UNDERLINED, 125
PANEL.VALUE_X, 125
PANEL.VALUE_Y, 125
PATH_IS_DIRECTORY, 125
PATH_LAST_VALIDATED, 125
PATH_RELATIVE_TO, 126
PATH_USE_FRAME, 126
SCREEN_NUMBER, 126
SCREEN_SERVER, 126
SCROLLBAR_COMPUTE_SCROLL_PROC, 126
SCROLLBAR_DIRECTION, 127
SCROLLBAR_LAST_VIEW_START, 127
SCROLLBAR_MENU, 127
SCROLLBAR_MOTION, 127
SCROLLBAR_NORMALIZE_PROC, 128
SCROLLBAR_NOTIFY_CLIENT, 128
SCROLLBAR_OBJECT_LENGTH, 128
SCROLLBAR_PAGE_LENGTH, 128
SCROLLBAR_PIXELS_PER_UNIT, 129
SCROLLBAR_SPLITTABLE, 129
SCROLLBAR_VIEW_LENGTH, 129
SCROLLBAR_VIEW_START, 129
SEL_APPEND_TYPE_NAMES, 129
SEL_APPEND_TYPES, 130
SEL_CONVERT_PROC, 130
SEL_COPY, 130
SEL_DATA, 131
SEL_DONE_PROC, 131
SEL_FIRST_ITEM, 131
SEL_FORMAT, 132
SEL_LENGTH, 132
SEL_LOSE_PROC, 132
SEL_., 136
SEL_NEXT_ITEM, 132
SEL_REQ_BYTESIZE, 275
SEL_REQ_COMMIT_PENDING_DELETE, 129
SEL_REQ_CONTENTS_ASCII, 276
SEL_REQ_CONTENTS_PIECES, 276
SEL_REQ_DELETE, 276
SEL_REQ_END_REQUEST, 276
SEL_REQ_FAKE_LEVEL, 276
SEL_REQ_FILE_NAME, 276
SEL_REQ_FIRST, 276
SEL_REQ_FIRST_UNIT, 277
SEL_REQ_LAST, 277
SEL_REQ_LAST_UNIT, 277
SEL_REQ_LEVEL, 277
SEL_REQ_RESTORE, 277
SEL_REQ_SET_LEVEL, 277
SEL_REQ_YIELD, 277
SEL_.OWN, 132
SEL_PROP_DATA, 133
SEL_PROP_FORMAT, 133
SEL_PROP_INFO, 133
SEL_PROP_LENGTH, 133
SEL_PROP_TYPE, 133
SEL_PROP_TYPE_NAME, 133
attribute (cont'd)

SEL_RANK, 134
SEL_RANK_NAME, 134
SEL_REPLY_PROC, 134
SEL_TIME, 135
SEL_TIMEOUT_VALUE, 135
SEL_TYPE, 135
SEL_TYPE_INDEX, 135
SEL_TYPE_NAME, 136
SEL_TYPES, 136
SERVER_ATOM, 136
SERVER_ATOM_NAME, 137
SERVER_EXTENSION_PROC, 137
SERVER_EXTERNAL_XEVENT_MASK, 137
SERVER_EXTERNAL_XEVENT_PROC, 138
SERVER_IMAGE_BITMAP_FILE, 138
SERVER_IMAGE_BITS, 138
SERVER_IMAGE_CMS, 139
SERVER_IMAGE_COLORMAP, 139
SERVER_IMAGE_DEPTH, 139
SERVER_IMAGE_PIXMAP, 140
SERVER_IMAGE_SAVE_PIXMAP, 140
SERVER_IMAGE_X_BITS, 140
SERVER_NTH_SCREEN, 140
SERVER_SYNC, 140
SERVER_SYNC_AND_PROCESS_EVENTS, 140
TEXTSW_ACTION.*, 141
TEXTSW_ACTION_CAPS_LOCK, 289
TEXTSW_ACTION_CHANGED_DIRECTORY, 289
TEXTSW_ACTION_CHANGED_DIRECTORY, 140
TEXTSW_ACTION贽EDITED_FILE, 289
TEXTSW_ACTION贽EDITED_MEMORY, 289
TEXTSW_ACTION贽FILE_IS_READONLY, 290
TEXTSW_ACTION贽LOADED_FILE, 290
TEXTSW_ACTION贽OPERATION, 290
TEXTSW_ACTION赟TOOL_DESTROY, 290
TEXTSW_ACTION赟TOOL_MGR, 290
TEXTSW_ACTION赟TOOL-fashion, 290
TEXTSW_ACTION赟USING_MEMORY, 290
TEXTSWAGAIN赟RECORDING, 141
TEXTSWАвтоIndent, 141
TEXTSWАвтоScroll_by, 141
TEXTSW Blink_Caret, 141
TEXTSW Browsing, 141
TEXTSW_CHECKPOINT_FREQUENCY, 141
TEXTSW_CLIENT_DATA, 142
TEXTSW_CONFIRM_OVERTWRITE, 142
TEXTSW_CONTENTS, 142
TEXTSW_CONTROL_CHARS_USE_FONT, 142
TEXTSW_DESTROY_VIEw, 142
TEXTSWDISABLE赟VIEW, 142
TEXTSW_DISABLE_LOAD, 142
TEXTSW_EDIT_COUNT, 143
TEXTSW_EXTRAS_CMD_MENU, 143
TEXTSW_FILE, 143
TEXTSW_FILE_CONTENTS, 143
TEXTSW_FIRST, 143
TEXTSW_FIRST_LINE, 143
TEXTSW_FONT, 25, 143
TEXTSW_HISTORY_LIMIT, 144
TEXTSW_IGNORE_LIMIT, 144
TEXTSW_INSERT_FROM_FILE, 144
TEXTSW_INSERTION_POINT, 144
TEXTSW_INSERT_MAKES_VISIBLE, 144
TEXTSW_LENGTH, 144
TEXTSW_LINE_BREAK_ACTION, 145
TEXTSW_LOWER_CONTEXT, 145
TEXTSW_MEMORY_MAXIMUM, 145
TEXTSW_MODIFIED, 145
TEXTSW_MULTI_CLICK_SPACE, 145
TEXTSW_MULTICLICK_TIMEOUT, 146
TEXTSW_NOTIFY_PROC, 146
TEXTSW_READ_ONLY, 146
TEXTSW_STATUS, 146
TEXTSW_STORE_CHANGED_FILE, 146
TEXTSW_SUBMENU_EDIT, 147
TEXTSW_SUBMENU_FILE, 147
TEXTSW_SUBMENU_FIND, 147
TEXTSW_SUBMENUViewSet, 147
TEXTSW_SUBMENUViewSet, 147
TEXTSW_UPPER_CONTEXT, 147
TTY_ARGV, 147
TTY_CONSOLE, 147
TTY_PAGE_MODE, 148
TTY_PID, 148
TTYQUIT_ON_CHILD_DEATH, 148
TTY_TTY_FD, 148
WIN_ALARM, 148
WIN_ALARM_DATA, 148
WINBACKGROUND_COLOR, 149
WINBACKGROUND_PIXMAP, 149
WIN_BELOW, 149
WIN_BIT_GRAYITY, 149
WIN_BORDER, 149
WIN_CLIENT_DATA, 149
WIN_CMD_LINE, 150
WIN_CMS, 150
WIN_CMS_DATA, 150
WIN_CMS_NAME, 150
WIN_COLLAPSE_EXPOSURES, 151
WIN_COLUMN_GAP, 151
WIN_COLUMNS, 151
WIN_COLUMN_WIDTH, 151
WIN_CONSUME_EVENT, 151
WIN_CONSUME_EVENTS, 151
attribute (cont’d)
WIN_CONSUME_X_EVENT_MASK, 152
WIN_CURSOR, 152
WIN_DEPTH, 152
WIN_DYNAMIC_VISUAL, 152
WIN_EVENT_PROC, 152
WIN_FIT_HEIGHT, 153
WIN_FIT_WIDTH, 153
WIN_FOREGROUND_COLOR, 153
WIN_FRAME, 153
WIN_FRONT, 153
WIN_GRAB_ALL_INPUT, 153
WIN_HORIZONTAL_SCROLLBAR, 154
WIN_IGNORE_EVENT, 154
WIN_IGNORE_EVENTS, 154
WIN_IGNORE_X_EVENT_MASK, 154
WIN.INHERIT_COLORS, 154
WIN_INPUT_MASK, 155
WIN_INPUT_ONLY, 155
WIN_IS_CLIENT_PANE, 155
WIN,KBD_FOCUS, 155
WIN_MAP, 155
WIN_MENU, 155
WIN_MESSAGE_DATA, 155
WIN_MESSAGE_FORMAT, 156
WIN_MESSAGE_TYPE, 156
WIN_MOUSE_XY, 156
WIN_NO_CLIPPING, 156
WIN_PARENT, 156
WIN_PERCENT_HEIGHT, 156
WIN_PERCENT_WIDTH, 156
WIN_RECT, 156
WIN.RETAINED, 157
WIN.RIGHT_OF, 157
WIN_ROW_GAP, 157
WIN_ROW_HEIGHT, 157
WIN_ROWS, 157
WIN/save, UNDER, 157
WIN_SCREEN_RECT, 158
WIN_SET_FOCUS, 158
WIN_SOFT_FNKEY_LABELS, 158
WIN_TOP_LEVEL, 158
WIN_TOP_LEVEL_NO_DECOR, 159
WIN_TRANSPARENT, 159
WIN_VERTICAL_SCROLLBAR, 159
WIN_WINDOW_GRAVITY, 159
WIN_COLOR, INDICES, 159
WIN_EVENT_MASK, 159
XV_APP_NAME, 160
XV_AUTO_CREATE, 160
XV_BOTTOM_MARGIN, 160
XV_DEPTH, 160
XV_DISPLAY, 160
XV_ERROR_PROC, 161
XV_FOCUS_ELEMENT, 161
XV_FONT, 161
XV_HEIGHT, 161
XV_HELP_DATA, 161
XV_INIT_ARGC_PTR_ARGV, 162
XV_INIT_ARGS, 162
XV_INSTANCE_NAME, 162
XV_KEY_DATA, 163
XV_KEY_DATA_REMOVE, 163
XV_KEY_DATA_REMOVE_PROC, 164
XV_LABEL, 164
XV_LC_BASIC_LOCALE, 164
XV_LC_DISPLAY_LANG, 165
XV_LC_INPUT_LANG, 165
XV_LC_NUMERIC, 165
XV_LC_TIME_FORMAT, 165
XV_LEFT_MARGIN, 166
XV_LOCALE_DIR, 166
XV_MARGIN, 166
XV_NAME, 166
XV_OWNER, 166
XV_RECT, 166
XV_RIGHT_MARGIN, 167
XV_ROOT, 167
XV_SCREEN, 167
XV_SHOW, 167
XV_TOP_MARGIN, 167
XV_TYPE, 167
XV_USAGE_PROC, 168
XV_USE_DB, 168
XV_USE_LOCALE, 169
XV_VISUAL, 169
XV_VISUAL_CLASS, 170
XV_WIDTH, 170
XV_X_ERROR_PROC, 170
XV_XID, 171
XV_Y, 171
ATTR_LIST attribute, 32

C

canvas, paint window, 7
subclassed from openwin, 7
subwindow, 7
view window, 7
CANVAS_AUTO_CLEAR attribute, 32
CANVAS_AUTO_EXPAND attribute, 32
CANVAS_AUTO_SHRINK attribute, 32
CANVAS_CMS_REPAINT attribute, 32
CANVAS_EACH_PAINT_WINDOW() macro, 175
CANVAS_END_EACH macro, 175
CANVAS_FIXED_IMAGE attribute, 32
CANVAS_HEIGHT attribute, 33
CANVAS_MIN_PAINT_HEIGHT attribute, 33
CANVAS_MIN_PAINT_WIDTH attribute, 33
CANVAS_NO_CLIPPING attribute, 33
CANVAS_NTH_PAINT_WINDOW attribute, 33
CANVAS_PAINT_CANVAS_WINDOW attribute, 34
CANVAS_PAINT_VIEW_WINDOW attribute, 34
CANVAS_PAINT_WINDOW_ATTRS attribute, 34
CANVAS_REPAINT_PROC attribute, 35
CANVAS_RESIZE_PROC attribute, 35
CANVAS_RETAINED attribute, 35
CANVAS_VIEWABLE_RECT attribute, 36
CANVAS_VIEW_CANVAS_WINDOW attribute, 36
CANVAS_VIEW_PAINT_WINDOW attribute, 36
CANVAS_WIDTH attribute, 36
CANVAS_X_PAINT_WINDOW attribute, 36
CHECKBOX_ITEM attribute, 17
CHOICE_ITEM attribute, 17
abbreviated, 17
checkbox, 17
exclusive and nonexclusive, 17
CMS_BACKGROUND_PIXEL attribute, 37
CMS_COLOR_COUNT attribute, 37
CMS_CONTROLS attribute, 37
CMS_CONTROL_CMS attribute, 37
CMS_FOREGROUND_PIXEL attribute, 38
CMS_INDEX attribute, 38
CMS_INDEX_TABLE attribute, 38
CMS_NAME attribute, 38
CMS_NAMED_COLORS attribute, 38
CMS_PIXEL attribute, 38
CMS_SCREEN attribute, 39
CMS_SIZE attribute, 39
CMS_TYPE attribute, 39
CMS_X_COLORS attribute, 39
cursor, hotspot, 8
cursor_background_color attribute, 39
cursor_copy() procedure, 175
cursor_drag_state attribute, 39
cursor_drag_type attribute, 40
cursor_foreground_color attribute, 40
cursor_image attribute, 40
cursor_mask_char attribute, 40
cursor_op attribute, 40
cursor_src_char attribute, 41
cursor_string attribute, 41
cursor_xhot attribute, 41
cursor_yhot attribute, 41

D
defaults_exists() procedure, 175
defaults_get_boolean() procedure, 176
defaults_get_character() procedure, 176
defaults_get_enum() procedure, 177
defaults_get_integer() procedure, 177
defaults_get_integer_check() procedure, 177
defaults_get_string() procedure, 177
defaults_init_db() procedure, 178
defaults_load_db() procedure, 178
defaults_lookup() procedure, 178
defaults_set_boolean() procedure, 178
defaults_set_character() procedure, 178
defaults_set_integer_check() procedure, 178
defaults_set_string() procedure, 179
defaults_store_db() procedure, 179
DND_ACCEPT_CURSOR attribute, 41
DND_ACCEPT_X_CURSOR attribute, 42
DND_CURSOR attribute, 42
dnd_decode_drop() function, 179
dnd_done() function, 180
dnd_is_forwarded() macro, 180
dnd_is_local() macro, 180
dnd_send_drop() function, 180
DND_TIMEOUT_VALUE attribute, 42
DND_TYPE attribute, 42
dnd_x_cursor attribute, 42
drop_target_item, 18
DROP_SITE_DEFAULT attribute, 43
DROP_SITE_DELETE_REGION attribute, 43
DROP_SITE_DELETE_REGION_PTR attribute, 43
DROP_SITE_EVENT_MASK attribute, 43
DROP_SITE_ID attribute, 43
DROP_SITE_REGION attribute, 44
DROP_SITE_REGION_PTR attribute, 44

E
event_action() macro, 180
event_alt_is_down() macro, 180
event_button_is_down() macro, 181
event_ctrl_is_down() macro, 181
event-driven input handling, 26
event_id() macro, 181
event_is_ascii() macro, 181

298 XView Reference Manual
event_is_button() macro, 181
event_is_down() macro, 181
event_is_iso() macro, 181
event_is_key_bottom() macro, 181
event_is_key_left() macro, 182
event_is_key_right() macro, 182
event_is_key_top() macro, 182
event_is_meta() macro, 182
event_is_string() macro, 182
event_is_up() macro, 182
event_left_is_down() macro, 182
event_meta_is_down() macro, 182
event_middle_is_down() macro, 182
event_right_is_down() macro, 182
events, Event structure, 5
event_shift_is_down() macro, 182
event_string() macro, 182
event_time() macro, 183
event_window() macro, 183
evnet_xevent() macro, 183
event_xevent_type() macro, 183
exclusive and nonexclusive choices, 17

F

FILE_CHOOSER_ABBREV_VIEW attribute, 44
FILE_CHOOSER_APP_DIR attribute, 44
FILE_CHOOSER_AUTO_UPDATE attribute, 45
FILE_CHOOSER_CD_FUNC attribute, 45
FILE_CHOOSER_CHILD attribute, 45
FILE_CHOOSER_COMPARE_FUNC attribute, 46
FILE_CHOOSER_CUSTOMIZE_OPEN attribute, 46
FILE_CHOOSER_DIRECTORY attribute, 47
FILE_CHOOSER_DOC_NAME attribute, 47
FILE_CHOOSER_EXTEN_FUNC attribute, 47
FILE_CHOOSER_EXTEN_HEIGHT attribute, 48
FILE_CHOOSER_FILTER_FUNC attribute, 48
FILE_CHOOSER_FILTER_MASK attribute, 49
FILE_CHOOSER_FILTER_STRING attribute, 49
FILE_CHOOSER_HISTORY_LIST attribute, 49
FILE_CHOOSER_MATCH_GLYPH attribute, 50
FILE_CHOOSER_MATCH_GLYPH_MASK attribute, 50
FILE_CHOOSER_SAVE_TO_DIR attribute, 51
FILE_CHOOSER_SHOW_DOT_FILES attribute, 51
FILE_CHOOSER_TYPE attribute, 51
FILE_CHOOSER_UPDATE attribute, 51
FILE_LIST_ABBREV_VIEW attribute, 52
FILE_LIST_AUTO_UPDATE attribute, 52
FILE_LIST_CHANGE_DIR_FUNC attribute, 52
FILE_LIST_COMPAR_FUNC attribute, 53
FILE_LIST_DIRECTORY attribute, 53
FILE_LIST_DOTDOT_STRING attribute, 53
FILE_LIST_FILTER_FUNC attribute, 54
FILE_LIST_FILTER_MASK attribute, 55
FILE_LIST_FILTER_STRING attribute, 53
FILE_LIST_MATCH_GLYPH attribute, 55
FILE_LIST_MATCH_GLYPH_MASK attribute, 55
FILE_LIST_MATCH_ROW_TYPE attribute, 55
FILE_LIST_ROW_TYPE attribute, 55
FILE_LIST_SHOW_DIR attribute, 56
FILE_LIST_SHOW_DOT_FILES attribute, 56
FILE_LIST_UPDATE attribute, 56
FILE_LIST_USE_FRAME attribute, 56
font conventions (in this book), boldface, xii
italic, xii
typewriter font, xii
font family, (See fonts).
font ID, 9
FONT_CHAR_HEIGHT attribute, 56
FONT_CHAR_WIDTH attribute, 57
FONT_DEFAULT_CHAR_HEIGHT attribute, 57
FONT_DEFAULT_CHAR_WIDTH attribute, 57
FONT_FAMILY attribute, 57
FONT_INFO attribute, 57
FONT_NAME attribute, 58
FONT_PIXFONT attribute, 58
FONT_RESCALE_OF attribute, 58
fonts, family, 9
scale, 9
size, 9
style, 9
using xv_find, 10
FONT_SCALE attribute, 58
FONT_SIZE attribute, 59
FONT_SIZES_FOR_SCALE attribute, 59
FONT_STRING_DIMS attribute, 60
FONT_STYLE attribute, 60
frame, subframe, 11
FRAME package, capabilities, 10
FRAME_ACCELERATOR attribute, 60
FRAME_BACKGROUND_COLOR attribute, 61
FRAME_BUSY attribute, 61
FRAME_CLOSED attribute, 61
FRAME_CLOSED_RECT attribute, 61
function (cont’d)
notify_remove_itimer_func(), 194
notify_remove_output_func(), 194
notify_remove_signal_func(), 194
notify_remove_wait3_func(), 194
notify_set_destroy_func(), 195
notify_set_event_func(), 195
notify_set_exception_func(), 195
notify_set_input_func(), 195
notify_set_itimer_func(), 196
notify_set_output_func(), 196
notify_set_signal_func(), 197
notify_set_wait3_func(), 197
notify_start(), 197
notify_stop(), 198
notify_veto_destroy(), 198

G

gauge item, 19
GC, font ID, 9
generic, functions; table of, 6
Generic Object, 5
glyphs, 9

H

handle, for object, 5
header file, icon.h, 12
server.h, 25
HELP_STRING_FILENAME attribute, 72
HISTORY_ADD_FIXED_ENTRY attribute, 73
HISTORY_ADD_ROLLING_ENTRY attribute, 73
HISTORY_DUPLICATE_LABELS attribute, 73
HISTORY_FIXED_COUNT attribute, 73
HISTORY_INACTIVE attribute, 74
HISTORY_LABEL attribute, 74
HISTORY_MENU_HISTORY_LIST attribute, 74
HISTORY_MENU_OBJECT attribute, 74
HISTORY_NOTIFY_PROC attribute, 74
HISTORY_ROLLING_COUNT attribute, 75
HISTORY_ROLLING_MAXIMUM attribute, 75
HISTORY_VALUE attribute, 75

I

icon, creating, 12
ICON_FONT attribute, 75
icon.h header file, 12
ICON_HEIGHT attribute, 75
ICON_IMAGE attribute, 76
ICON_IMAGE_RECT attribute, 76
ICON_LABEL attribute, 76
ICON_LABEL_RECT attribute, 76
ICON_MASK_IMAGE attribute, 76
ICON_TRANSPARENT attribute, 76
ICON_TRANSPARENT_LABEL attribute, 77
ICON_WIDTH attribute, 77
inheritance, 5, 7
input, handling; event-driven, 26;
mainline, 26
interclient communication, 24
item, gauge, 19

M

macro, CANVAS_EACH_PAINT_WINDOW(), 175
CANVAS_END_EACH, 175
dnd_is_forwarded(), 180
dnd_is_local(), 180
event_action(), 180
event_alt_is_down(), 180
event_button_is_down(), 181
event_ctrl_is_down(), 181
event_id(), 181
event_is_ascii(), 181
event_is_button(), 181
event_is_down(), 181
event_is_iso(), 181
event_is_key_bottom(), 181
event_is_key_left(), 181
event_is_key_right(), 181
event_is_key_top(), 182
event_is_meta(), 182
event_is_string(), 182
event_is_up(), 182
event_left_is_down(), 182
event_meta_is_down(), 182
event_middle_is_down(), 182
event_right_is_down(), 182
event_shift_is_down(), 182
event_string(), 182
event_time(), 183
event_window(), 183
event_xevent(), 183
event_xevent_type(), 183
macro (cont’d)
MENUITEM_SPACE, 183
OPENWIN_END_EACH, 198
PANEL_END_EACH, 199
rect_borderadjust(), 200
rect_bottom(), 200, 202
rect_construct(), 201
rect_equal(), 201
rect_includespoint(), 201
rect_includesrect(), 201
rect_intersectrect(), 202
rect_isnull(), 202
rect_passtochild(), 203
rect_passtoparent(), 203
rect_print(), 203
rect_right(), 203
rect_sizes_differ(), 203
window_fit(), 210
window_fit_height(), 210
window_fit_width(), 210
XV_DISPLAY_FROM_WINDOW(), 212
mainline input handling, 26
MENU_ACCELERATOR attribute, 77
MENU_ACTION_ACCELERATOR attribute, 77
MENU_ACTION_IMAGE attribute, 78
MENU_ACTION_ITEM attribute, 78
MENU_APPEND_ITEM attribute, 78
MENU_CLASS attribute, 78
MENU_CLIENT_DATA attribute, 78
MENU_COL_MAJOR attribute, 79
MENU_COLOR attribute, 78
MENU_DEFAULT attribute, 79
MENU_DEFAULT_ITEM attribute, 79
MENU_DESCEND_FIRST attribute, 79
MENU_DONE_PROC attribute, 79
menu_done_proc() procedure, 79
MENU_FEEDBACK attribute, 80
MENU_FIRST_EVENT attribute, 80
MENU_GEN_PIN_WINDOW attribute, 80
MENU_GEN_PROC attribute, 80
menu_gen_proc() procedure, 80-81
MENU_GEN_PROC_IMAGE attribute, 80
MENU_GEN_PROC_ITEM attribute, 81
MENU_GEN_PULLRIGHT attribute, 81
MENU_GEN_PULLRIGHT_IMAGE attribute, 81
MENU_GEN_PULLRIGHT_ITEM attribute, 81
MENU_IMAGE attribute, 81
MENU_IMAGE_ITEM attribute, 81
MENU IMAGES attribute, 81
MENU_INACTIVE attribute, 82
MENU_INSERT attribute, 82
MENU_INSERT_ITEM attribute, 82
MENU_ITEM attribute, 82
MENUITEM_SPACE macro, 183
MENU_LAST_EVENT attribute, 82
MENU_NCOLS attribute, 83
MENU_NITEMS attribute, 83
MENU_NOTIFY_PROC attribute, 83
menu_notify_proc() procedure, 83
MENU_NOTIFY_STATUS attribute, 83
MENU_NROWS attribute, 84
MENU_NTH_ITEM attribute, 84
MENU_PARENT attribute, 84
MENU_PIN attribute, 84
MENU_PIN_PROC attribute, 84
menu_pin_proc() procedure, 84
MENU_PIN_WINDOW attribute, 85
MENU_PULLRIGHT attribute, 85
MENU_PULLRIGHT_IMAGE attribute, 85
MENU_PULLRIGHT_ITEM attribute, 85
MENU_RELEASE attribute, 85
MENU_RELEASE_IMAGE attribute, 85
MENU_REMOVE attribute, 85
MENU_REMOVE_ITEM attribute, 86
MENU_REPLACE attribute, 86
MENU_REPLACE_ITEM attribute, 86
menu_release_item() procedure, 183
menu_return_value() procedure, 184
MENU_SELECTED attribute, 86
MENU_SELECTED_ITEM attribute, 86
menu_show() procedure, 184
MENU_STRING attribute, 86
MENU_STRING_ITEM attribute, 86
MENU_STRINGS attribute, 87
MENU_STRINGS_AND_ACCELERATORS attribute, 87
MENU_TITLE attribute, 87
MENU_TITLE_ITEM attribute, 87
MENU_TYPE attribute, 87
MENU_VALID_RESULT attribute, 88
MENU_VALUE attribute, 88
message item, 20
multiline text item, 21-22

N

NOTICE_BLOCK_THREAD attribute, 88
NOTICE_BUSY_FRAMES attribute, 88
NOTICE_BUTTON attribute, 88
NOTICE_BUTTON_NO attribute, 88
NOTICE_BUTTON_YES attribute, 89
NOTICE_EVENT_PROC attribute, 89
NOTICE_FOCUS_XY attribute, 89
NOTICE_FONT attribute, 90
procedure (cont’d)
defaults_set_integer_check(), 178
defaults_set_string(), 179
defaults_store_db(), 179
frame_get_rect(), 183
frame_set_rect(), 183
menu_done_proc(), 79
menu_gen_proc(), 80-81
menu_notify_proc(), 83
menu_pin_proc(), 84
menu_return_item(), 183
menu_return_value(), 184
menu_show(), 184
notice_prompt(), 184
OPENWIN_EACH_VIEW(), 198
openwin_split_destroy_proc(), 93
panel_advance_caret(), 198
panel_background_proc(), 95
panel_backup_caret(), 198
PANEL_CHECK_BOX, 198
PANEL_CHOICE_STACK, 199
PANEL_EACH_ITEM(), 199
panel_event_proc(), 101
panel_paint(), 199
panel_text_notify(), 200
PANEL_TOGGLE, 200
rect_below(), 200
rect_bounding(), 200
rect_clipvector(), 201
rect_distance(), 201
rect_intersection(), 202
rect_order(), 202
rect_right_of(), 203
SCROLLABLE_PANEL, 204
scrollbar_default_compute_scroll_proc(), 126
scrollbar_paint(), 204
sel_convert_proc(), 130, 204
sel_done_proc(), 131
selection_*, 204
selection_acquire(), 281
selection_ask(), 281
selection_clear_functions(), 282
selection_create(), 282
selection_destroy(), 282
selection_done(), 282
selection_figure_response(), 283
selection_get_file(), 283
selection_inform(), 283
selection_init_request(), 283
selection_inquire(), 284
selection_inquire_all(), 284
selection_query(), 284
selection_report_event(), 285
selection_request(), 285
selection_yield_all(), 285
sel_lose_proc(), 132
sel_post_req(), 205
sel_reply_proc(), 134
textsw_add_mark(), 205
textsw_append_file_name(), 205
textsw_delete(), 205
textsw_edit(), 205
textsw_erase(), 205
textsw_find_bytes(), 206
textsw_find_file(), 206
textsw_find_mark(), 206
textsw_insert(), 207
textsw_match_bytes(), 207
textsw_next(), 207
textsw_normalize_view(), 207
textsw_notify_proc(), 146
textsw_possibly_normalize(), 207
textsw_remove_mark(), 208
textsw_replace_bytes(), 208
textsw_reset(), 208
textsw_save(), 208
textsw_screen_line_count(), 208
textsw_scroll_lines(), 209
textsw_set_selection(), 209
textsw_store_file(), 209
tty_output(), 209
window_done(), 210
window_read_event(), 210
win_set_kbd_focus(), 210
wmgr_completechangerect(), 211
wmgr_draw(), 211
wmgr_update(), 211
wmgr_completechangerect(), 211
wmgr_open(), 211
wmgr_refreshwindow(), 211
wmgr_set_top(), 211
txv_col(), 212
txv_cols(), 212
txv_create(), 212
txv_destroy(), 212
txv_error(), 213
txv_error_format(), 213
txv_find(), 213
txv_get(), 213
txv_init(), 214
txv_row(), 214
txv_rows(), 214
XV_SCREEN_FROM_WINDOW(), 215
procedure (cont’d)
  xv_send_message(), 215
  XV_SERVER_FROM_WINDOW(), 215
  xv_set(), 215
  xv_super_set_avlist(), 216
  xv_unique_key(), 216
  xv_usage(i), 216
  xv_window_loop(), 216
  xv_window_return(), 217

R
Rect structure, 5
rect_below() procedure, 200
rect_borderadjust() macro, 200
rect_bottom() macro, 200
rect_bound() procedure, 201
rect_construct() macro, 201
rect_distance() procedure, 201
rect_equal() macro, 201
rect_includespoint() macro, 201
rect_intersection() procedure, 202
rect interpolate() macro, 202
rect_isnull() macro, 202
Rectlist structure, 5
rect_order() procedure, 202
rect_passtochild() macro, 203
rect_passstoparent() macro, 203
rect_print() macro, 203
rect_right() macro, 203
rect_right_of() procedure, 203
rect_sizes_differ() macro, 203

S
SCREEN_NUMBER attribute, 126
SCREEN_SERVER attribute, 126
SCROLLABLE_PANEL procedure, 204
scrollbar, about, 24
SCROLLBAR_COMPUTE_SCROLL_PROC
  attribute, 126
scrollbar_default_compute_scroll_proc() pro-
  cedure, 126
SCROLLBAR_DIRECTION attribute, 127
SCROLLBAR_LAST_VIEW_START attribute,
  127
SCROLLBAR_MENU attribute, 127
SCROLLBAR_MOTION attribute, 127
SCROLLBAR_NORMALIZE_PROC attribute,
  128
SCROLLBAR_NOTIFY_CLIENT attribute, 128
SCROLLBAR_OBJECT_LENGTH attribute, 128
SCROLLBAR_PAGE_LENGTH attribute, 128
scrollbar_paint() procedure, 204
SCROLLBAR_PIXELS_PER_UNIT attribute,
  129
SCROLLBAR_SPLITTABLE attribute, 129
SCROLLBAR_VIEW_LENGTH attribute, 129
SCROLLBAR_VIEW_START attribute, 129
scrolling lists, 20
SEL_APPEND_TYPE_NAMES attribute, 129
SEL_APPEND_TYPES attribute, 130
SEL_CONVERT_PROC attribute, 130
sel_convert_proc() procedure, 130, 204
SEL_COPY attribute, 130
SEL_DATA attribute, 131
SEL_DONE_PROC attribute, 131
sel_done_proc() procedure, 131
selection, about, 24
selection_* procedure, 204
selection_acquire() procedure, 281
selection_ask() procedure, 281
selection_clear_functions() procedure, 282
selection_create() procedure, 282
selection_destroy() procedure, 282
selection_done() procedure, 282
selection_figure_response() procedure, 283
selection_init_request() procedure, 283
selection_inquire() procedure, 284
selection_inquire_all() procedure, 284
selection_query() procedure, 284
selection_report_event() procedure, 285
selection_request() procedure, 285
selection_yield_all() procedure, 285
SEL_FIRST_ITEM attribute, 131
SEL_FORMAT attribute, 132
SEL_LENGTH attribute, 132
SEL_DONE_PROC attribute, 132
sel_done_proc() procedure, 132
SELN_attribute, 136
SEL_NEXT_ITEM attribute, 132
SELN_REQ_BYTESIZE attribute, 275
SELN_REQ_COMMIT_PENDING_DELETE
  attribute, 275
SELN_REQ_CONTENTS_ASCII attribute, 276
SELN_REQ_CONTENTS_PIECES attribute, 276
SELN_REQ_DELETE attribute, 276
SELN_REQ_END_REQUEST attribute, 276
SELN_REQ_FAKE_LEVEL attribute, 276
Index

TEXTSW_EXTRAS_CMD_MENU attribute, 143
TEXTSW_FILE attribute, 143
TEXTSW_FILE_CONTENTS attribute, 143
textsw_file_lines_visible() procedure, 206
textsw_find_bytes() procedure, 206
textsw_find_mark() procedure, 206
TEXTSW_FIRST attribute, 143
textsw_first() procedure, 206
TEXTSW_FIRST_LINE attribute, 143
TEXTSW_FONT attribute, 25, 143
TEXTSW_HISTORY_LIMIT attribute, 144
TEXTSW_IGNORE_LIMIT attribute, 144
textsw_index_for_file_line() procedure, 206
textsw_insert() procedure, 207
TEXTSW_INSERT_FROM_FILE attribute, 144
TEXTSW_INSERTION_POINT attribute, 144
TEXTSW_INSERT_MAKES_VISIBLE attribute, 144
TEXTSW_LENGTH attribute, 144
TEXTSW_LINE_BREAK_ACTION attribute, 145
textsw_match_bytes() procedure, 207
TEXTSW_MEMORY_MAXIMUM attribute, 145
TEXTSW_MODIFIED attribute, 145
textsw_normalize_view() procedure, 207
TEXTSW_MULTICLICK_SPACE attribute, 145
TEXTSW_MULTICLICK_TIMEOUT attribute, 146
textsw_next() procedure, 207
textsw_normalize_view() procedure, 207
TEXTSW_NOTIFY_PROC attribute, 146
textsw_notify_proc() procedure, 146
textsw_possibly_normalize() procedure, 207
TEXTSW_READ_ONLY attribute, 146
textsw_remove_mark() procedure, 208
textsw_replace_bytes() procedure, 208
textsw_reset() procedure, 208
textsw_save() procedure, 208
textsw_screen_line_count() procedure, 208
textsw_scroll_lines() procedure, 209
textsw_set_selection() procedure, 209
TEXTSW_STATUS attribute, 146
TEXTSW_STORE_CHANGES_FILE attribute, 146
textsw_store_file() procedure, 209
TEXTSW_SUBMENU_EDIT attribute, 147
TEXTSW_SUBMENU_FILE attribute, 147
TEXTSW_SUBMENU_FIND attribute, 147
TEXTSW_SUBMENU_VIEW attribute, 147
TEXTSW_UPPER_CONTEXT attribute, 147
TTY_ARGV attribute, 147
TTY_CONSOLE attribute, 147
TTY_PAGE_MODE attribute, 148
TTY_PID attribute, 148
TTY.Quit_ON_CHILD.DEATH attribute, 148
ttysw_input() procedure, 209
ttysw_output() procedure, 209
TTY.TTY_FD attribute, 148
type, about, 5
Xv.object, 5
Xv_opaque, 5
U
user interface standards, 3
V
view window, 7
W
WIN_ALARM attribute, 148
WIN_ALARM_DATA attribute, 148
WIN_BACKGROUND_COLOR attribute, 149
WIN_BACKGROUND_PIXMAP attribute, 149
WIN_BELOW attribute, 149
WIN_BIT_GRAVITY attribute, 149
WIN_BORDER attribute, 149
WIN_CLIENT_DATA attribute, 149
WIN_CMD_LINE attribute, 150
WIN_CMS attribute, 150
WIN_CMS_DATA attribute, 150
WIN_CMS_NAME attribute, 150
WIN_COLLAPSE_EXPOSURES attribute, 151
WIN_COLUMN_GAP attribute, 151
WIN_COLUMNS attribute, 151
WIN_COLUMN_WIDTH attribute, 151
WIN_CONSUME_EVENT attribute, 151
WIN_CONSUME_EVENTS attribute, 151
WIN_CONSUME_X_EVENT_MASK attribute, 152
WIN_CURSOR attribute, 152
WIN_DEPTH attribute, 152
window_done() procedure, 210
window_fill() macro, 210
window_fill_height() macro, 210
window_fill_width() macro, 210
window_read_event() procedure, 210
WIN_DYNAMIC_VISUAL attribute, 152
WIN_EVENT_PROC attribute, 152
WIN_FIT_HEIGHT attribute, 153
WIN_FIT_WIDTH attribute, 153
WIN_FOREGROUND_COLOR attribute, 153
WIN_FRAME attribute, 153
WIN_FRONT attribute, 153
WIN_GRAB_ALL_INPUT attribute, 153
WIN_HORIZONTAL_SCROLLBAR attribute, 154
WIN_IGNORE_EVENT attribute, 154
WIN_IGNORE_EVENTS attribute, 154
WIN_IGNORE_X_EVENT_MASK attribute, 154
WIN_INHERIT_COLORS attribute, 154
WIN_INPUT_MASK attribute, 155
WIN_INPUT_ONLY attribute, 155
WIN_IS_CLIENT_PANE attribute, 155
WIN_KBD_FOCUS attribute, 155
WIN_MAP attribute, 155
WIN_MENU attribute, 155
WIN_MESSAGE_DATA attribute, 155
WIN_MESSAGE_FORMAT attribute, 156
WIN_MOUSE_XY attribute, 156
WIN_NO_CLIPPING attribute, 156
WIN_PARENT attribute, 156
WIN_PERCENT_HEIGHT attribute, 156
WIN_PERCENT_WIDTH attribute, 156
WIN_RECT attribute, 156
WIN_RETAINED attribute, 157
WIN_RIGHT_OF attribute, 157
WIN_ROW_GAP attribute, 157
WIN_ROWS attribute, 157
WIN_SAVE_UNDER attribute, 157
WIN_SCREEN_RECT attribute, 158
WIN_SET_FOCUS attribute, 158
WIN_SOFT_FNKEY_LABELS attribute, 158
WIN_TOP_LEVEL attribute, 158
WIN_TOP_LEVEL_NO_DECOR attribute, 159
WIN_TRANSPARENT attribute, 159
WIN_VERTICAL_SCROLLBAR attribute, 159
WIN_WINDOW_GRAVITY attribute, 159
WIN_X_COLOR_INDICES attribute, 159
WIN_X_EVENT_MASK attribute, 159
wmgr_bottom() procedure, 210
wmgr_changelevel() procedure, 211
wmgr_close() procedure, 211
wmgr_completerehangerect() procedure, 211
wmgr_open() procedure, 211
wmgr_refreshwindow() procedure, 211
wmgr_top() procedure, 211

X

X Window System, interclient communications, 24
XV_APP_NAME attribute, 160
XV_AUTO_CREATE attribute, 160
XV_BOTTOM_MARGIN attribute, 160
xv_col() procedure, 212
xv_cols() procedure, 212
xv_create() procedure, 212
XV_DEPTH attribute, 160
xv_destroy() procedure, 212
XV_DISPLAY attribute, 160
XV_DISPLAY_FROM_WINDOW() macro, 212
xv_error() procedure, 213
xv_error_format() procedure, 213
XV_ERROR_PROC attribute, 161
xv_find() procedure, 213
XV_FOCUS_ELEMENT attribute, 161
XV_FONT attribute, 161
xv_get() procedure, 213
XV_HEIGHT attribute, 161
XV_HELP_DATA attribute, 161
XView, generic functions, 6
Generic Object, 5
xv_init() procedure, 214
XV_INIT_ARGC_PTR_ARGV attribute, 162
XV_INIT_ARGS attribute, 162
XV_INSTANCE_NAME attribute, 162
XV_KEY_DATA attribute, 163
XV_KEY_DATA_REMOVE attribute, 163
XV_KEY_DATA_REMOVE_PROC attribute, 164
XV_LABEL attribute, 164
XV_LC_BASIC_LOCALE attribute, 164
XV_LC_DISPLAY_LANG attribute, 165
XV_LC_INPUT_LANG attribute, 165
XV_LC_NUMERIC attribute, 165
XV_LC_TIME_FORMAT attribute, 165
XV_LEFT_MARGIN attribute, 166
XV_LOCALE_DIR attribute, 166
XV_MARGIN attribute, 166
XV_NAME attribute, 166
XV_OBJECT, 5
Xv_object type, 5
Xv_opaque type, 5
XV_OWNER attribute, 166
XV_RECT attribute, 166
XV_RIGHT_MARGIN attribute, 167
XV_ROOT attribute, 167
xv_row() procedure, 214
xv_rows() procedure, 214
XV_SCREEN attribute, 167
XV_SCREEN_FROM_WINDOW() procedure, 215
xv_send_message() procedure, 215
XV_SERVER_FROM_WINDOW() procedure, 215
xv_set() procedure, 215
XV_SHOW attribute, 167
xv_super_set_avlist() procedure, 216
XV_TOP_MARGIN attribute, 167
XV_TYPE attribute, 167
xv_unique_key() procedure, 216
xv_usage() procedure, 216
XV_USAGE_PROC attribute, 168
XV_USE_DB attribute, 168
XV_USE_LOCALE attribute, 169
XV_VISUAL attribute, 169
XV_VISUAL_CLASS attribute, 170
XV_WIDTH attribute, 170
xv_window_loop() procedure, 216
xv_window_return() procedure, 217
XV_X, 170
XV_X_ERROR_PROC attribute, 170
XV_XID attribute, 171
XV_Y attribute, 171
About the Editor

Thomas van Raalte has a degree in Computer Science from the University of Vermont. He lives in Portland, Oregon and runs a consulting company, Computer Rooster, specializing in technical writing.