sun ®

microsystems

XView Version 2 Reference Manual: Converting SunView Applications

Part No: 800-4836-10
Revision A, of 11 June 1990

Contents

Tables

Figures

Preface

This manual describes how to convert your SunView™ programs to XView™ programs. This manual is most effective when used in conjunction with the *XView Programming Manual*, and other X manuals from O'Reilly and Associates (including the *Xlib Programming Manual* and *Xlib Reference Manual*). In addition, example XView programs demonstrating various XView objects are provided in the following directory:

```
$OPENWINHOME/share/src/sun/xview/examples
```

XView Definition

The XView toolkit is an object-oriented user interface toolkit designed for the X Window System™ version 11 (including X11/NeWS™).[1] Based upon the highly popular SunView toolkit, the XView toolkit has been designed and extended to take advantage of the features of a networked window system while maintaining much of the flavor and the Application Programmer's Interface (API) of its predecessor. The XView toolkit is portable across many versions of the UNIX® operating system as well as other operating systems.[2] It provides an implementation of the **OPEN LOOK™ Graphical User Interface Specification.**[3]

Converting to XView Format

There are two conversion scenarios: for a quick, *minimal conversion*, and for a *full conversion*. The minimal conversion lets you convert a program quickly by retaining a number of SunView features and facilities that may not be supported (or supported with the same implementation) in future XView releases. A full conversion requires a few more changes

1. The X Window System is a trademark of the Massachusetts Institute of Technology.
2. UNIX is a registered trademark of AT&T.
3. OPEN LOOK is a trademark of AT&T; UNIX is a registered trademark of AT&T.

now, but it reduces the number of changes that may be required with subsequent releases.

The current release tape includes a set of `sed` scripts (called by a shell script) to help you convert with minimum effort. However, while these scripts can save you considerable time and effort, they cannot do everything. There are instructions for using these scripts in Chapter 2.

Planning Your Conversion Effort

Every effort has been made to automate the conversion process. However, depending upon your individual SunView application, you will need to spend some time and effort on completing the process. You will find that some applications will convert almost entirely automatically. With others, you will need to put more work in to a final, converted program.

Here are some general guidelines that may help you to gauge the amount of time it will take to convert your existing program:

- If your program uses any of the low-level functionality described in the *SunView 1 System Programmer's Guide*, you will find that this functionality will need to be converted manually. One rule of thumb to use when contemplating a conversion project is to assume that the more heavily you rely on low-level routines, the more time you should budget to complete the conversion.
- If your program includes any routines that predate SunView from the SunOS releases prior to release 3.0, or if it includes any low-level SunWindows features, you will find that you must redesign portions of your code to accommodate the fact that functionality at this level is now the responsibility of the X11 (or X11/NeWS) server, or has been removed completely.
- Some Pixrect routines are now implemented by the server, or are not supported at all. If you use any of these routines, you will need to rework those portions of your code.
- If your application uses a lot of graphics and has been tuned for performance in a kernel-based window system, you will probably need to redesign the graphics portion to tune it for performance in a server-based window system. You can still have your application converted and running in a short time, but the re-tuning may take somewhat longer.

If You Have SunView Programming Experience

One of the major XView design objectives was to preserve as much compatibility as possible between SunView and XView applications. If you are an experienced SunView programmer, this manual should contain most of the information you need to convert an application. Converting SunView programs to XView format is a largely mechanical and not very difficult task, since the two Application Programmer's Interfaces (API) are similar. Experienced SunView programmers will also want to consult the other XView manuals, to learn more about specific aspects of the XView API.

If You Do Not Have SunView Programming Experience

To convert a SunView program to XView format without SunView experience you will definitely want to study this document. It will provide you with most of what you need to know, and further information will be available shortly. The series of manuals on the X Window System published by O'Reilly & Associates are highly recommended; and, if you have access to a copy of the *SunView 1 Programmer's Guide*, you will probably want to keep it handy, too.

How This Manual is Organized

The contents of this manual are as follows:

- Chapter 1 presents an architectural overview of the XView toolkit.
- Chapter 2 discusses issues to consider prior to converting SunView to XView. It also describes the automated conversion procedures.
- Chapter 3 provides conversion instructions categorized by object and package.
- Chapter 4 is a comparative listing of SunView and XView attributes.
- Chapter 5 introduces new features.
- Appendix A lists defaults. It also contains a man page for the conversion script `convert_to_Xdefaults`.
- Appendix B contains hints for improving performance.
- Appendix C consists of a table of cross references called by the conversion script `convert_to_xview`, and a man page for the script.
- Appendix D contains a global listing of changes in reserved prefixes from SunView to XView.
- Appendix E discusses Pixwin to Xlib procedures.
- Appendix F discusses debugging techniques.

Typographical Conventions

Body text is set in Times Roman, with *italics* for emphasis. Attributes, procedures, macros, and anything resembling C code are all set in `Courier`, using the following C conventions:

Procedures are lower case and are followed by parentheses, as in

```
xv_get()
```

Macros are all upper case and are followed by parentheses, as in the following example, except where they are lower case in source code.

```
OPENWIN_EACH_VIEW()
```

Attributes are all upper case but are not followed by parentheses, as in

```
CANVAS_MIN_PAINT_HEIGHT
```

Representations of anything that might appear on your screen are set in `Courier`. In addition, code examples are set in round-cornered boxes to set them off from the surrounding text; when these boxes contain user type-in, their surrounding round-cornered boxes are shaded as well.

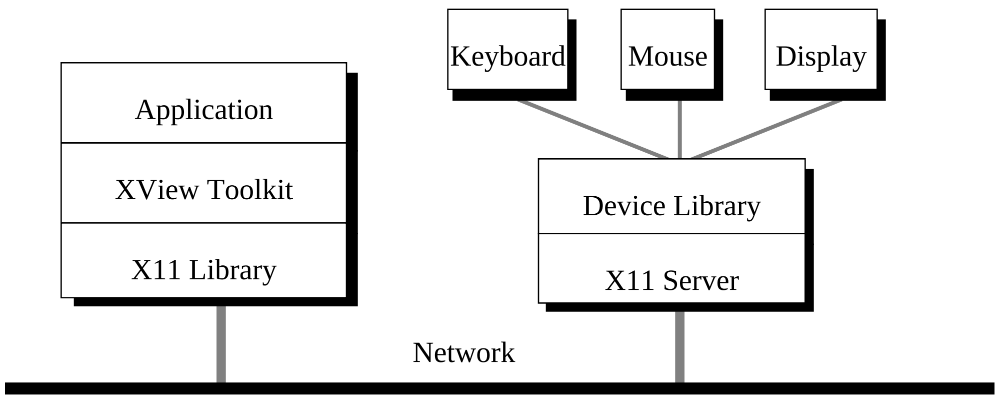
XView Overview

The XView Toolkit is an objected-oriented user interface toolkit designed for the X Window System version 11 (X11). As currently configured on Sun workstations, it runs on the ‘‘X’’ side of the X11/NeWS server. XView functionality is based on the mature SunView toolkit, so it not only provides familiar, comprehensive functionality, but also offers a sensible migration path for existing SunView applications. This overview describes XView’s architecture and contrasts it with that of SunView. For more in-depth information on XView architecture as well as creating applications in XView, refer to the first several chapters in the XView Programming Manual from O’Reilly and Associates.

1.1 System Architecture

XView system architecture differs markedly from SunView’s. XView is implemented for a server-based window system (X11) while SunView was implemented for a kernel-based window system (SunWindows).

Figure 1-1 *System Structure: an XView application running on a network*



The basic difference between these two system architectures is that kernel-based systems are usually hardware and operating system specific, while server-based systems use a protocol that is independent of hardware and operating systems. Server-based window systems can run on one machine and display their output in windows on another machine anywhere on the network, regardless of machine architecture, operating systems, display resolutions, and color capabilities.

1.2 XView Application Architecture

Application architecture is identical in XView and in SunView. Each toolkit provides a framework for assembling the pre-built user interface components with application-specific code. But instead of simply providing programmers with a library of procedures they can call to *build* applications, XView provides a framework for defining and then *managing* applications.

In XView, the application programmer uses variable-length attribute-value lists based on C's *varargs* to specify objects to be created, such as windows, menus, and scrollbars. Since the usual behavior for each object is already defined, the programmer only need specify deviations from the default behavior. With a typical procedural interface, on the other hand, the programmer must completely describe the object being specified. (While procedural interfaces generally result in applications filled with boiler-plate code, XView largely eliminates it.)

After specifying objects, the XView programmer defines call-back procedures, which the toolkit calls to notify the application of events or user actions. Finally, the programmer connects, or registers, the application-specific code with XView and lets XView manage the application.

With XView's attribute-value interface, calls are short when the default values are sufficient, and long—but understandable—when the developer needs to specify many values. In any case, the fact that the XView toolkit provides default values for the attributes frees the application programmer to specify *less* toolkit behavior.

1.3 XView Object Model

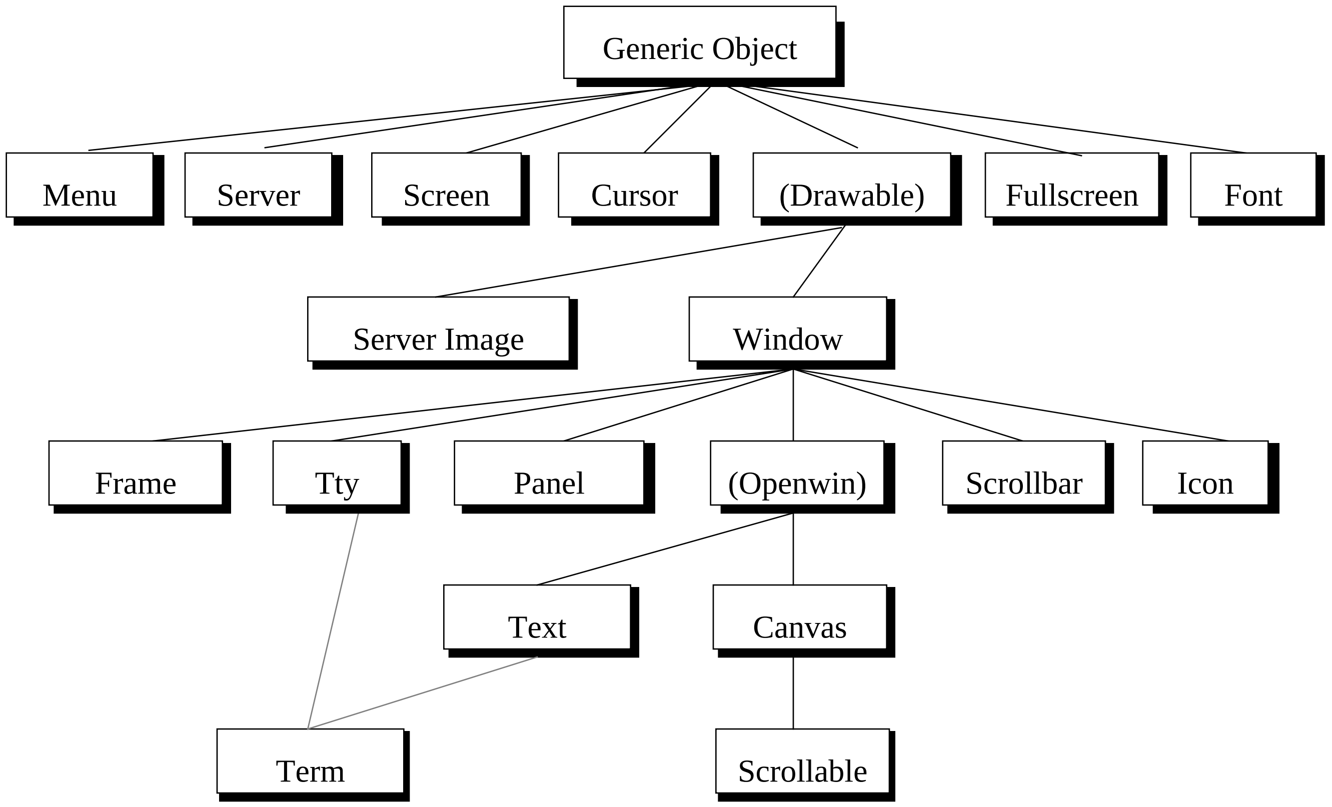
Since XView is an object-oriented system with static subclassing, a programmer can use building blocks such as windows, text, panels, and icons to construct an application. All XView objects share a common set of functions and retain SunView's facility for variable-length argument lists of attributes.

Object Class Hierarchy

XView's classes are implemented mostly as static leaf classes; there are almost no hidden classes without instances. Subclasses inherit attributes from their superclass, and, with one exception, all inheritance is single inheritance.

Figure 1-2 shows the XView class hierarchy and the relationships between the classes. For example, a *Panel* is a subclass of *Canvas*, which is a particular subclass of *Window*, which in turn is a particular subclass of the *Generic Object*. Objects that act as drawing surfaces are referred to as drawable objects, and are instances of subclasses of the *Drawable* class.

Figure 1-2 *XView Object Class Hierarchy*



Generic Objects, Common Attributes

All classes are subclasses of the Generic Object class. Generic objects implement all the features that allow objects to be created and destroyed, and provide some truly *generic attributes* (which take the prefix `XV_`), such as `XV_TYPE`, `XV_LABEL`, `XV_KEY_DATA`, and `XV_OWNER`.

A few attributes, such as `XV_X`, `XV_RIGHT_MARGIN`, and `XV_FONT`, are not actually attributes of the generic object, even though they have the generic `XV_` prefix. These attributes are *common* rather than universal, and there are some XView classes that do not support them. Attributes shared among objects are ordinarily named at the lowest level of commonality, usually at the generic level. When different objects have related properties, they are controlled by attributes of the same name; for example, `XV_FONT` is used instead of `WIN_FONT`, `PANEL_FONT`, and `MENU_FONT`.

Generic Procedures

XView also provides a set of generic procedures. Where possible, as with `Font` and `Server` objects, XView uses reference counting.

`xv_init()`	initializes the notifier; reads passed attributes; initializes, loads, and reads appropriate databases.
`xv_find()`	can be used instead to locate and share common objects; if unsuccessful, `xv_find()` reverts to `xv_create()`.
`xv_create()`	creates objects.
`xv_get()`	queries an object's attributes.
`xv_set()`	changes the object's attributes.
`xv_destroy()`	destroys an object when it is no longer needed

1.4 Window Objects

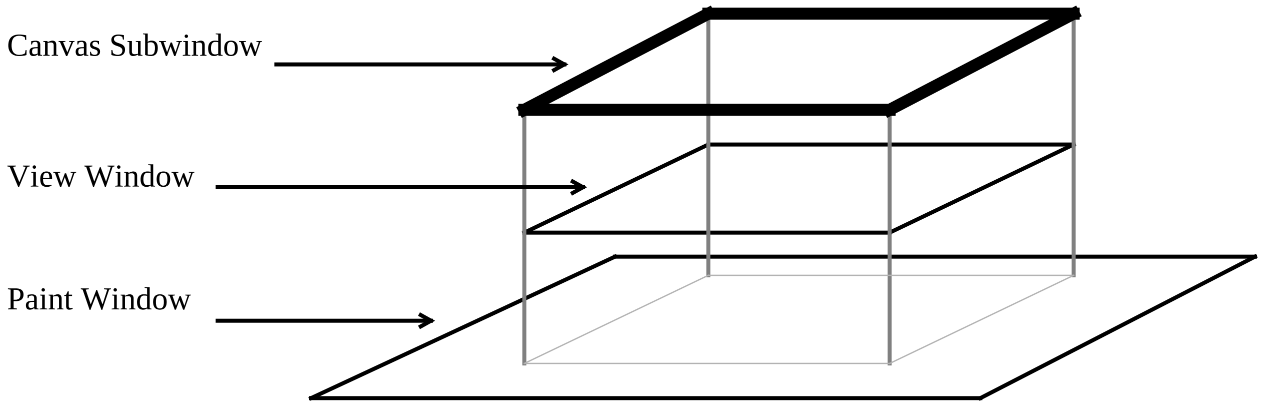
Window objects include frames and subwindows. Frames contain non-overlapping subwindows within their borders. Currently, there are six types of subwindow objects in XView:

Canvas subwindow	A subwindow into which programs can draw.
Panel subwindow	A control panel subwindow containing control objects.
Scrollbar	An object attached to and usually displayed with a subwindow, through which a user can control which portion of a subwindow's contents are displayed.
Term subwindow	A scrollable, editable terminal emulator.
Text subwindow	A subwindow containing text.
Tty subwindow	A terminal emulator in which commands can be given and programs executed.

Canvas Subwindows

The most basic feature of any toolkit is the ability to provide application programmers with a drawing surface. Here they can build applications with features not already in the toolkit or unique to the application. XView provides a canvas object that lets applications draw on an area larger than the visible window where the drawing appears.

Figure 1-3 *The XView Canvas Model*



While the SunView drawing surface is implemented as a Pixwin region, the XView drawing surface is a separate window, the *paint window*. The paint window is clipped to another window, the *view window*, so that only the part of the paint window ‘‘underneath’’ the view window shows through. The view window appears, together with optional scrollbars, in the actual canvas subwindow. The result is a much cleaner implementation than in SunView, both conceptually and programmatically.

Panels and Menus

SunView panels and menus are separate packages, offering almost no overlapping functional or visual consistency, while in XView, control panels and menus are visually identical. Many of the objects available as panel items, such as buttons and choice-items, are also available as menu items. Menus are no longer only transient objects. They can be pinned†, and thereafter behave like panels.

In XView, panels are implemented as windows, while menus and individual panel items and menu items in them are not. There are two reasons for this approach:

- SunView windows are ‘‘heavy-weight’’ objects, in which every window is represented by a UNIX file descriptor. They have to be used sparingly as there is an OS-defined limit as to how many file descriptors can be created per process. XView preserves the tradition of sparing use of windows.
- Windows are not that much ‘‘lighter-weight’’ in X11 (as XView objects that connect to the server use up file descriptors and

†Pinning a menu is the OPEN LOOK UI concept of clicking on a visual push-pin in an object to make it stay on the screen. See the ‘‘OPEN LOOK Graphical User Interface Specification’’ for more information.

there is some performance considerations) so complex panels and menus containing large numbers of items would sustain poor performance if every item were a window.

The panel or menu, acting as a mini-notifier, can deal with the distribution of events to the individual panel and menu item objects better than can an implementation where every item is a window. A scrollable panel is a panel on which a scrollbar can be attached.

Text Subwindows

A text editing area is another basic component of any toolkit. The XView text subwindow retains most of the features of the SunView text subwindow, except that it now uses the OPEN LOOK UI selection model. In XView, the text subwindow's contents are stored in the client process; the server only displays the text, it does not store or manipulate it.

As in SunView (since SunOS 4.0), the text subwindow is a full-featured text editor that supports the ISO Latin-1 character set. It provides:

- checkpointing (automatic save after so many keystrokes)
- keyboard-control of the insertion point (in addition to using the mouse)
- both memory and file editing
- arbitrary numbers of Undo and Again actions
- find-and-replace pop-up windows
- a customizable user-specific menu

As a result of implementing the OPEN LOOK User Interface, the text subwindow also provides the wipe-through-pending-delete text selection and editing model. Additionally, the drag/move editing accelerator allows direct manipulation of selected text.

Scrollbars

In XView, scrollbars are windows that implement the OPEN LOOK UI scrollbar metaphor of an elevator on a cable. The scrollbar package only manages its scrollbar window. It is responsible for keeping the elevator and proportional indicator up to date, but it is the responsibility of the window attached to the scrollbar to update the window's contents. For example, the text subwindow package is responsible for scrolling the text.

In SunView, scrollbars are implemented on Pixwins, rather than on windows, because of the limited number of windows available to an application. (Every window is represented by a file-descriptor, and there are only so many per process).

Tty and Terminal Emulation Subwindows

A common building-block for any UNIX-based toolkit is a tty/shell emulation capability. XView provides both a basic shell subwindow package (`TTY`) and a more user-friendly, scrollable shell subwindow package (`TERM`). The `TTY` package provides a standard UNIX shell in an XView subwindow. The `TERM` package is a derivative of the `TTY` package, which is subclassed from both the tty and text subwindow classes. `TERM` subwindows are scrollable and include most of the same OPEN LOOK UI editing features found in the text subwindow.

In XView, as in SunView, the shell is a separate process that runs concurrently with the client program. The implementation of this parent-child model assures the user's ability to choose the underlying shell program (C-Shell, Borne Shell, Korn Shell, and so on) while continuing to run the same subwindow package on top of it.

1.5 Openwin

Openwin is a new package that provides XView window pane management services. Openwin clients register with Openwin that they are going to represent some unknown type of data (such as WYSIWYG text or raster images). Openwin then manages the application program's windows. Openwin also supports *split* and *join* of window panes. Thus, a user can split the viewing area into multiple viewing areas, each of which represents the same data in a different view, without the need for application program intervention. The text, term, and canvas subwindows are implemented as subclasses of Openwin.

1.6 Other Visual Objects

There are other types of visual objects displayed on the screen that differ from the subwindows described above in that they are less general and serve more specific functions. They include:

Panel item	An object in a control panel that allows interaction between the user and the application. Panel items can be moved, displayed and undisplayed. There are several predefined types of items, including buttons, choice items (both exclusive and non-exclusive), message items, text items, and sliders. Panel items are not implemented as windows; instead they use the panel's event dispatcher to receive input events.
Notice	A box on the screen that informs the user of some condition that requires attention. It usually has one or more buttons which the user can push to confirm, deny (if a decision is required), or continue.
Cursor	A shape on the screen representing the user's point of attention, controlled by the mouse.

Icon	A window representing an application when the application's frame is in the ‘‘closed’’ state.

Cursors

XView cursors have a similar function to SunView cursors, and cursor shape varies according to window. However, in XView, the method by which cursors are implemented differs from SunView. In SunView, cursors are memory pixrects, while in XView, *cursor objects* use *server image objects* to represent the image on the server. This is one area where the XView API is significantly different from the SunView API.

Icons

XView icons are, as in SunView, pictures representing an application in a closed state. In SunView, the icon is displayed by sizing the open window to the shape of the icon image and painting the image into that (usually) smaller window. In XView, the icon is manipulated by the window manager. When the window manager determines that the icon should be shown (for example, when the user presses a "Close" button in the window header), it queries that window's properties for the icon to display. XView sets these properties for its applications. This is another area where the XView API is different from the SunView API.

1.7 Non-Visual Objects

Non-visual objects cannot be drawn into or displayed directly, but are still subclassed from the generic object:

Server	The XView front-end for an X11 server connection. The window-server is the program that actually does the drawing for XView and any other X11 client programs. It also receives the user's mouse and keyboard input. The server may run on a different computer, which in turn may run under a different operating system, than the one the XView application is running on. An XView application can also connect to more than one server.
Screen	The actual screen on which the program's drawing/ output appears. Some computers have multiple screens, of different resolutions and color capabilities.
Font	The font used when drawing text. Different servers have different fonts available. XView can use any font available to X11 clients. For forward compatibility purposes, the otherwise opaque font object is type-compatible with a Pixfont. The bits that represent the font images are stored on the server, not with the XView application.

1.8 Imaging Facilities

In SunView, the Pixwin interface is responsible for imaging. Pixwin is a logical layer over the Pixrect library. It implements the basic windowing notions of *clipping* and *coexistence* with other windows. In XView, imaging is carried out by the X11 window server, through whatever graphical imaging implementation it uses.

In SunView, Pixrect provides Pixwin with a hardware-abstracted library of graphical operations for the display of images. Pixrect supports the concepts of screen coordinates, pixels, bitmaps, raster operations, vectors, and text drawing.

In XView, Xlib provides the hardware-abstracted library of graphical operations. Because the actual implementation of the graphical routines is in the server, the graphics can be implemented for any hardware or operating system. The X11 library provides the same concepts of screen coordinates, pixels, bitmaps, and assorted drawing operations as Pixwin. To provide seamless transition for SunView programmers, the Pixwin interface of SunView was retained, but the implementation was discarded and replaced by X11 graphical operations. Pixwin calls are therefore only wrappers to the underlying Xlib calls, and it is recommended that you make Xlib calls directly.

Server Images

In SunView, all images are pixrects, which are manipulated either as data structures in memory (called *memory pixrects*) or as device files that represent the ability to access a display device. When a pixrect is ready to be displayed, the bits representing the image already reside on the same machine as the display hardware that will display it. This is the optimal way of handling the data, since the image data resides where it is needed.

In XView however, the window server may potentially run on a different machine or even a different architecture. It is even possible that the client and the server represent data altogether differently. In order to store the bits of an image on the window server, and thus avoid the need to ship the same data across the server connection repeatedly, XView provides *server image objects*, which take advantage of the data format conversion provided in the X11 protocol. A server image object is an X11 Pixmap, which is represented on the client-side as a pixrect. This approach allows server images to be treated as pixrects, maintaining SunView compatibility. The client program manipulates server images in the same way it manipulates other XView objects.[†]

[†]With `xv_create(), xv_get(), xv_set()`, and so on.

1.9 The XView Notifier

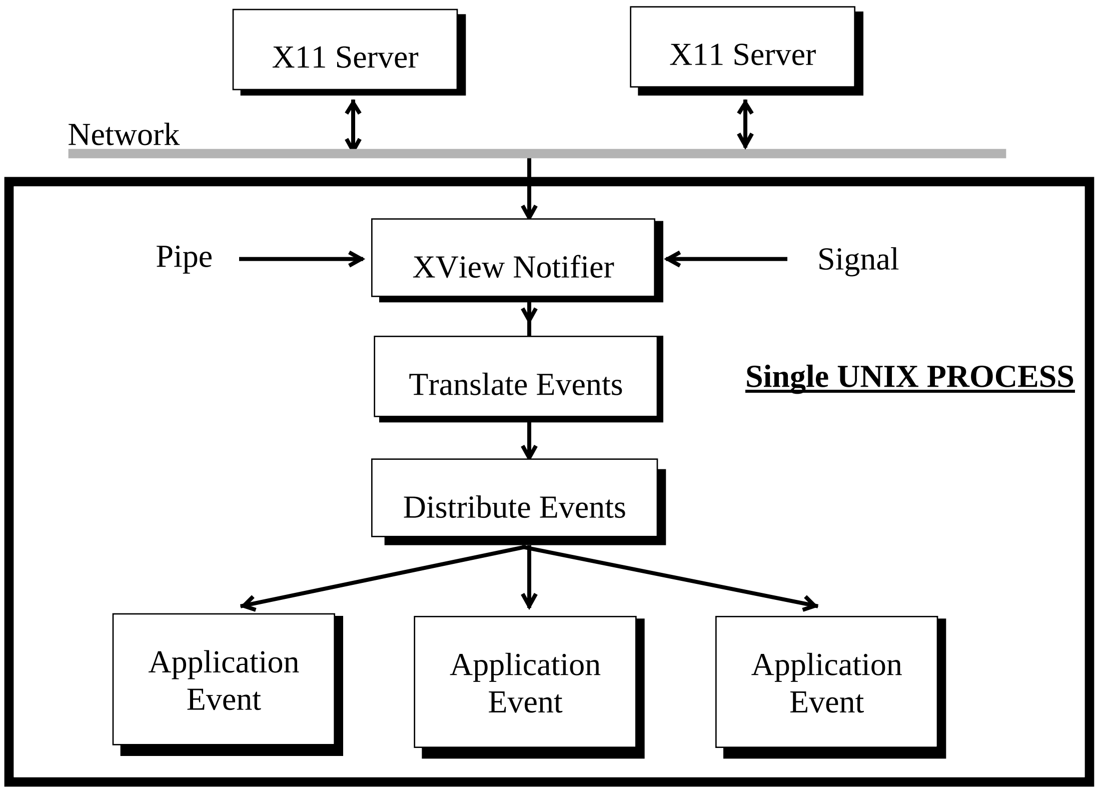
The SunView Notifier facility was converted from the kernel-based SunWindows input environment to X11 with few outward signs of change.

In SunView, every window is represented as a file descriptor, while in XView, a single file descriptor is used to connect to the server. In SunView, windows are registered with the Notifier, which waits for input on the file descriptor for each window and sends the input to the appropriate package. In XView, windows are also registered, but the Notifier waits for input on the file descriptor that is connected to the server. X11 input events are then translated to XView semantic events, such as `ACTION_SELECT`, `ACTION_PASTE`, `ACTION_STOP`, and then sent to the appropriate package. In other words, the XView Notifier has been unplugged from the kernel and its dependence on file-descriptors, then plugged into the X11 input stream without disturbing the basic implementation.

In X11, events are asynchronous data generated by the server as a result of some device activity or, possibly, as a side-effect of a request from an Xlib function. Events are dispatched to a client program whose window's event mask matches the event being processed. The XView Notifier reads the server connection by waiting for input on the UNIX socket connected to the X11 window server. If the application is displaying on two or more different machines, the Notifier waits for input on more than one socket. As in SunView, an application can also register other file descriptors (such as a pipe) or certain UNIX signals with the Notifier.

The Notifier weaves events from all these sources into a single, ordered event stream. This event stream eliminates the need for the application to poll separate streams from the different devices. Because the underlying Notifier multiplexes the input stream between windows, each individual window operates under the illusion that it has the user's full attention. That is, it sees precisely those events that the user has directed to it. Figure 1-4 shows the relationship of the XView Notifier to multiple X11 servers. It also shows the flow of events through the XView Notifier to the application's event procedures.

Figure 1-4 *The XView Notifier*



1.10 Font Handling

The use of fonts in SunView is generally unorganized and left to the discretion of the application programmer. There are no symbolic relationships (such as family, style, size) among the different fonts other than those their file names imply (such as `screen.r.12`, `screen.r.14`). Various different fonts are provided with SunView, and other fonts can easily be created with a font editing tool, called `fontedit`. SunView fonts are manipulated through the Pixfont font routines, part of the Pixrect library.

XView requires a richer font manipulation environment for application programmers. The standard X11 fonts are not guaranteed to have any properties, and the interpretations of the property values still have to be derived from a prior knowledge of the property. Though these properties added a few interesting pieces of information when compared with Pixfonts, they are still not quite what XView application programmers will need. XView font objects have the following new attributes:

Family The name of the font-family (such as Times, `Courier`, Helvetica)

Style	The style of this font in the family (such as **Bold**, *Italic*, ***Bold Italic Strike Underline***)
Size	The pixel size of the font (such as 10, 11, 12, 14, 18)
Scale	The relationship of this font's size to similar fonts in the same family/style group (such as small, medium, large, extra-large)
Rescale	This property allows a font to be *scaled* from one pixel size to another within a family/style group (such as from the small font to the large font).

XView font objects effectively replace the standard X11 font properties and routines. Fonts are discussed further in Chapter 3.

1.11 Color

Using Color in XView is radically different from using color in SunView. We will say a few things about color in the following paragraphs, but we recommend that you refer to the XView Programming Manual for details on implementing color in your application programs.

XView applications deal with color by allocating colormap segments. A colormap segment is a subset of the available cells in a colormap. Colormap segments can be allocated either *dynamic* (read-write) or *static* (read-only).

You can change the color cells in a dynamic colormap segment. The cells in a static colormap segment are initialized once and are read-only from then onwards, although it is not necessary for all the cells to be initialized at the same time. The XView library will try to return the closest (or exact) matching color cells from the X11 server.

(If the X11 server supports `StaticColor` as the default, the closest matching color will be returned. If the server supports `PseudoColor` as the default, the returned colormap cell may be allocated in one of several ways. If a previously allocated read-only cell matches the requested color, it is returned. If an exact match is not found and free unallocated cells are available, a cell is allocated and initialized with the required RGB values. If there are no free cells and an exact match is not found, an error is returned.)

Applications must always use static colormap segments unless they absolutely require read-write color cells. Static colormap segments, by sharing color cells across applications, use the shared hardware colormap resource more efficiently and reduce flashing.

Any object subclassed from the window object may allocate and use colormap segments. Such objects should implement the appropriate color inheritance model. For example, setting colors on a text subwindow will propagate the colors to all the views of the text subwindow. However, the colors can also be set on a particular view, as the following code fragments demonstrate

1.12 OPEN LOOK Graphical User Interface

The OPEN LOOK Graphical User Interface was chosen for XView because the OPEN LOOK UI will be the standard graphical interface for UNIX System V release 4. Besides offering programmers greatly enhanced functionality, it permits ‘‘look and feel’’ consistency with other OPEN LOOK UI applications.

Note that when you convert to XView you will not be able to customize your applications to look exactly like they did in SunView. There are restrictions that the toolkit adheres to in order to meet the Open Look Specifications, and SunView programmers need to consider the implications of this. Applications that use XView should try to adhere to the Open Look Graphical User Interface Application Style Guidelines published by Addison-Wesley. The following figure shows a sample OPEN LOOK screen.

Figure 1-5 *Sample OPEN LOOK UI applications*

1.13 Inter-client Communication

XView supports the ability to select objects on the screen, such as drawings or portions of text, and transfer the selection between applications. Selections within XView are managed in conjunction with the server according to a set of conventions referred to as the *Inter-Client Communications Conventions Manual* (ICCCM), which is a MIT Consortium Standard. Additionally, XView's interactions with an external window manager are defined by the ICCCM.

In SunView, a separate daemon process communicates with all selection service clients. In XView, the window server provides the same facility as the SunView selection service daemon. Each application program communicates with the server through an XView selection service interface, which is actually a front-end to XView's implementation of the ICCCM. The XView selection model is, like SunView, based upon the requestor/holder model of peer-to-peer communications.

The SunView window manager is built into SunView, and thus only SunView-style window management is provided. However, XView on X11 lets a user select a window manager. For maximal functionality and user interface consistency, an OPEN LOOK UI window manager that conforms to the guidelines of the ICCCM is the preferred XView window manager.

Another thing to note is that the window manager, which is a separate client application, controls the frame window and other decorations. The programmer has no control over it. Under the Open Look Window Manager, you can't change background or foreground color of the frame, or things like the frame menu. You can no longer do it programmatically from an application or the command line option, you can only use workspace properties to tell the window manager (olwm) what colors to set for the desktop.

1.14 Portability

XView was originally intended as a migration path for users of the SunView toolkit. During the process of implementing it on X11, however, there were relatively few problems in porting the toolkit to other hardware platforms. The main problem with porting was that SunView uses the Pixrect library, which is available only on Sun hardware. XView supports, but does not *require*, pixrects, and is implemented purely on top of X11. Thus few, if any, problems are anticipated in porting XView to other platforms that fully implement X11. In fact, XView is available from MIT's X Consortium, and a fairly large number of ports to a variety of platforms is expected shortly.

1.15 Summary

Most of the features in Sun's proprietary window system, SunWindows, are now available in X11, which has the substantial benefit of supporting distributed window applications. Also, the object-oriented system model yields a clean application programming interface. In SunView, this model was used in the text, panel and menu packages; in XView, the object-oriented system model, using C's varargs to support attribute-value lists, was extended to almost all packages. XView thus adds capabilities that were previously impossible, and is portable to non-Sun hardware.

Preliminary Considerations

SunView to XView conversion simply means taking SunView code and converting it to XView code. Once converted, a program can run on any X platform running XView. The converted XView program will work the same as it did in SunView with the exception of user interface differences. Unlike SunView, the appearance and functionality of XView follows the OPEN LOOK Graphical User Interface (GUI) specification. (Note that some SunView attributes have been included in XView for compatibility purposes, these are noted in later chapters.)

Once a program is converted to XView, it cannot run in a SunView environment. If you need to run a program in both XView and SunView environments, then you must maintain two source code versions. In general, you should keep the user-interface portion of the code separate from the application portions of the code.

2.1 Minimal or Full Conversion

Because some SunView code will still work in the current version of XView, different levels of SunView to XView conversion is possible. For example, in a minimal conversion you would only change those SunView routines and attributes not supported in XView. A minimal conversion is only recommended as a way to get an application running as soon as possible, and as a prelude to a full SunView to XView conversion.

Full conversion is recommended for performance and reliability reasons. Many of the SunView functions and attributes are only supported in XView for compatibility purposes. In the future, support for these functions and attributes may be eliminated. Note also that full conversion does not ensure 100% OPEN LOOK compliancy. To be 100% compliant you must refer to the OPEN LOOK UI Style Guide.

SunView Compatibility in XView

In some cases, a SunView feature is compatible with XView because only its name has changed. You can continue referring to such an attribute or procedure with the old SunView name, since it is aliased (via the **`#define`** statement) to the new XView name. Either name will produce the same functionality.

Other SunView features are still available for use in XView. You may choose to retain such attributes or procedures in your XView program when you convert. They will have the same effect as they had in SunView, which means they will not conform to the OPEN LOOK UI.

Another set of SunView attributes can still be used in XView even though their functionality has changed. In some cases, the functionality change may be minor, in others, extensive.

Finally, a few SunView features are not available in XView. Attempts to use these features will cause errors.

To convert your program fully to XView, you will have to make more changes than those needed for the minimal conversion. A SunView program is considered fully converted to XView if it contains no compatibility attributes or procedures.

Deciding Between a Full or Minimal Conversion

Making the correct decision regarding minimal or full conversion can save a lot of time and aggravation. We suggest the following guidelines:

- For low-investment applications, such as those developed rapidly or ad hoc for limited, in-house use, a minimal conversion is probably adequate. It may also be useful if you want to maintain a single copy of source code for both SunView and XView versions of your applications.
- For high-investment, real end-products, a full conversion is desirable. The cost, however, is that you cannot maintain SunView attributes and fonts under XView.

Note: When doing any conversion from SunView to XView, you will need to convert your Pixwin graphics function calls (pw_*) to Xlib graphics function calls. See Appendix E.

2.2 Automated Conversion

The first step in converting SunView to XView is to use the automated conversion tool, `convert_to_xview`. `convert_to_xview` is a shell script which uses **`sed`** to automate the conversion process. It takes a SunView file as its input, then goes through the file and replaces Sunview procedures, functions, and attributes with XView procedures, functions, and attributes. If an XView replacement is not possible, the script will flag that part of the code and direct you to the appropriate replacement package or function which you will then have to change by hand.

Note that running `convert_to_xview` is only the first step in converting an SunView file to XView. After running the conversion script, you will have to go through the file replacing SunView code (which the script could not automatically replace) with XView code. The following chapter describes this procedure in detail.

In addition to going through the converted file and replacing the SunView code, you may have to consider some architectural changes to the application itself. Even though the function of an application may remain the same, the XView user interface is different from the SunView user interface. For this reason you may have to change the way the function is presented to the user.

Step One: Setting the Path

Before performing automated conversion, make sure your `$OPENWINHOME` environmental variable is set correctly. Ideally, your `$PATH` should include `$OPENWINHOME/bin/xview`. In any case, the process of converting files is straightforward.

Invoking `convert_to_xview`

To invoke the shell script for full conversion on a program named *foo.c*, type:

```
% convert_to_xview foo.c
```

During processing, you should see the following:

```
  ---Converting file: foo.c
  --Done
```

After processing, the script will return output files with the extension, `.converted_to_xview`. This is shown below.

```
 foo.c.converted_to_xview
```

Note that multiple files could be converted in a similar manner:

```
% convert_to_xview foo.c bar.c
```

The script removes all backward-compatible attributes and makes as complete a transition to XView as possible. It will also flag code that needs to be changed by hand, and will reference the appropriate section or sections of this manual.

```
#ifdef XVIEW_COMMENT
/* XView CONVERSION -  Icons are now XView objects
that must be created, read Sect 4.3 on how to
convert to the new API
#ifdef XVIEW_COMMENT
```

For minimal conversion, type:

```
% convert_to_xview -m foo.c
```

After processing, the script will return a new output file with the extension, `.converted_to_xview`. This, just like the regular conversion. The difference, however, is that the script will maintain compatibility with the old SunView API wherever possible. All code that must be changed by hand will still be flagged and references to the appropriate section or sections of this manual cited. For instance:

```
#ifdef XVIEW_COMMENT
/*XView CONVERSION - Use new FULLSCREEN pkg
instead. See Sect 3.2 and 4.3 for replacement */
fullscreen_init(...);
#endif
```

Edit the Output File

After `convert_to_xview` is run, the next step, whether you are doing a full or a minimal conversion, is to edit the output file and make the changes flagged by the conversion script. From here, the mechanics are simple, although some architectural changes may be required. Simply search for the string ‘‘`XView CONVERSION`’’, then follow the adjacent comments such as:

```
base_frame =window_create((Xv_Window)NULL, FRAME,
                FRAME_ICON, my_icon,
                FRAME_LABEL, "foo",
#ifdef XVIEW_COMMENT
XView CONVERSION - Make sure to use xv_init to
process the attrs first. Sect 3.2
#endif
FRAME_ARGC_PTR_ARGV,   &argc, argv,
0);
```

The comments will either give you exact directions to follow or refer you to a particular section or subsection of this manual. Where appropriate, as above, there will be both directions and references.

Compiling XView Programs

Use the following command line to compile your XView programs:

```
% cc filename.c -lxview -lolgx -lX11 -o filename
```

If the OpenWindows libraries are not installed in **`/usr/lib`** and the OpenWindows header files are not installed in **`/usr/include,`** then you need to add the following command line options: **`-I$INCLUDEDIR`** and **`-L$LIBDIR`**. Note also that the SunOS shared library search path $LD_LIBRARY_PATH must be set appropriately. See the XView reference manual page for more information on environmental variables.

SunView to XView Conversion

This chapter describes the precise steps necessary for converting SunView applications to the XView. Note that there may be some overlap between the changes described in this section, and the changes performed by `convert_to_xview`.

Generally, each numbered section in this chapter describes the code changes required for an object or class. In other words, changes to the font code will be in the section entitled *Fonts*; changes to canvas code will be in the section called *Canvases*. Each section may also be subdivided into a subsection entitled *Compile-time Incompatibilities*, which are changes that will cause compile-time errors if not implemented, and *Run-time Incompatibilities*, which are changes that will cause run-time errors if not implemented.

This chapter is most effective when used in conjunction with the XView Programming Manual, and other X manuals from O'Reilly and Associates. In addition, example XView programs demonstrating various XView objects are provided in the following directory:

```
$OPENWINHOME/share/src/sun/xview/examples
```

3.1 Full Conversion Procedures

The changes described in this section are required for full XView compatibility. Note that these changes can be performed after all other changes described in this chapter are performed.

1) Convert all `window/panel/menu/scrollbar/...` `create()`, `set()`, `get()` and `destroy()` calls to `xv_create()`, `xv_set()`, `xv_get()`, and `xv_destroy()` calls. Note that you must remove all SunView features such as `ATTR_ROW`, which are supported in XView for compatibility with the SunView-style `window_create()`, `window_set()`, `panel_create_item()`, and `panel_set()` calls, but are not supported by the native XView `xv_*()` calls.

2) Convert `window_destroy()` calls to `xv_destroy_check()` calls. (In SunView, windows would not actually destroy themselves immediately; the `xv_destroy_check()` routine preserves those semantics).

3) While it was normal in SunView to create a scrollbar with a `NULL` parent, then attach it to a `textsw`, `panel` or `canvas`, the same procedure produces substantially poorer performance in XView. For better performance, do not create the scrollbar until the intended target is created, then create the scrollbar as a child of the target. For instance:

```
panel = xv_create(frame, PANEL, 0);
xv_create(panel, SCROLLBAR, 0);
```

4) Convert all create functions as follows:

```
xv_create(owner, package_name, attribute list,
        0);
```

Specifying `NULL` as an owner of non-window packages (scrollbars, menus, cursors, etc.) will work as long as your application initializes correctly and does not attempt to talk to multiple screens or servers.

5) Many attribute names have been redefined (‘‘aliased’’) with `#define` into common `XV_` attributes, as shown in Table 341

6) -1. For a complete listing of redefined attributes, refer to the relevant section in this chapter.

Table 3-1 *Some Attribute Names Collapsed Into Common XV_ Attributes*

SunView Attribute		***Common XV_ Attribute***
`ICON_FONT`	→	`XV_FONT`
`ICON_WIDTH`	→	`XV_WIDTH`
`MENU_FONT`	→	`XV_FONT`
`MENU_WIDTH`	→	`XV_WIDTH`
`MENU_LEFT_MARGIN`	→	`XV_LEFT_MARGIN`
`MENU_PARENT`	→	`XV_OWNER`
`SCROLL_RECT`	→	`XV_RECT`
`SCROLL_HEIGHT`	→	`XV_HEIGHT`
`SCROLL_WIDTH`	→	`XV_WIDTH`
`TEXTSW_LEFT_MARGIN`	→	`XV_LEFT_MARGIN`
`TEXTSW_MENU`	→	`XV_MENU`
`WIN_WIDTH`	→	`XV_WIDTH`
`WIN_LEFT_MARGIN`	→	`XV_LEFT_MARGIN`
`WIN_OWNER`	→	`XV_OWNER`
`FRAME_ARGC_PTR_ARGV`	→	`XV_ARGC_PTR_ARGV`
`FRAME_LABEL`	→	`XV_LABEL`

7) Change calls to `frame_cmdline_help(name)` or the old `tool_usage()` to `xv_usage(name)`. The new help facility requires you to provide help text in a separate file connected to your program with the `XV_HELP_DATA` attribute. To do help properly, you need to document most features at the graphical object level. (See the chapter on the Help package in the XView Programming Manual.)

8) `ATTR_ROW` and `ATTR_COL` work if you use `window_create()`, `window_set()`, `panel_create_item()`, or `panel_set()`, but not if you use `xv_create()`. Convert them to the new functions `xv_row()` or `xv_rows()` and `xv_col()` or `xv_cols()`. For example, the Sunview code shown below:

```
PANEL_ITEM_X   ATTR_COL(10)
PANEL_ITEM_Y   ATTR_ROW(1)
```

would be replaced by the following XView code.

```
XV_Y,  xv_row(1)
XV_X,  xv_col(10)
```

9) Remove all `pixwins` as drawing handles. Just pass the window returned by `xv_create()` into drawing routines.
10) Get rid of all use of window `FD`'s in routines. Just pass the object returned by `xv_create()`.
11) Call `xv_init()` explicitly at the beginning of your program and convert `FRAME_ARGC_PTR_ARGV` to `XV_ARGC_PTR_ARGV`.
12) Instead of calling `pf_open(font_name)`, call:

```
my_font = xv_find(window, FONT, FONT_NAME,
    font_name, 0);
```

Note: Since there is only one input mask in XView, be careful not to clobber it by setting the pick mask and then the keyboard mask.

13) Convert calls to `win_numbertoname()` and `win_fdtoname()` into `xv_get()` of `XV_XNAME.`
14) Convert `win_fdtonumber()` calls to `xv_get` or `XV_XID`.
15) Merge calls to `win_set()`, `win_get_kbd()`, and `win_pick_mask()` to `xv_set()` and `xv_get()` of `WIN_INPUT_MASK`.
16) Use `window_read_event()` instead of `input_readevent()`.

3.2 Miscellaneous Changes

The changes described in this section do not fall under specific object or class categories, however, they must be followed in order for your programs to work.

Include Files

Before a converted program can be compiled, be aware of, and perform the following changes to the `include` files:

1) The SunView master file, which includes most of the other `include` files, is `<suntool/sunview.h>`
   In XView, the master file is `<xview/xview.h>`
2) The `min` and `max` macros have been renamed `MIN` and `MAX.` They have been moved from `<sunwindow/sun.h>` to `<xview/base.h>.`

3) The definitions of `TRUE` and `FALSE` have been moved from `<sunwindow/rect.h>` to `<xview/base.h>`.

CAUTION: When restructuring `#include` lines, be careful if you change the order of included files. Many `<xview>` files depend on other files, such as `<stdio.h>`. These *must* be included first.

4) Change `#include` <`sunwindow/`*header_file*.`h`> and `#include <suntool/`*header_file*.`h>` to `#include <xview/`*header_file*.`h>`
5) Remove the following files: `<suntool/tool_struct.h>`, `<sunwindow/sun.h>`, and `<suntool/tool_hs.h>`
6) You may need to include `<xview/win_input.h>`.
7) You will need `<xview/cursor.h>` if your program uses cursors, and `<xview/icon.h>` if your program uses icons. In XView, you must create them with `xv_create()`, not with the `#DEFINE_FROM` macros used in SunView.

Declarations

This section describes declaration changes.

1) Change all `struct pixwin*` declarations to `Pixwin`. Accessing fields of the `struct` causes compile-time errors.
2) Change references to `struct cursor` *mycursor* to `Xv_Cursor` *mycursor*.
3) Change references to `struct icon` *myicon* to `Icon` *myicon*.
4) Do not attempt to access fields in `struct pixwin`, `struct icon` or `struct cursor`. Such attempts will produce compile time errors.
5) If you declare a variable as `struct cursor` *mycursor* in your application, your code may have calls that look something like `window_set(win, WIN_CURSOR, &mycursor).` These calls no longer work. Convert them to XView with `xv_set`. The same applies to `icon`.
6) If your program has references to `LINT_CAST` you need to remove them.

Default Calling Parameters

The default routines documented in the *SunView 1 System Programmers' Guide* have changed. The following default calling parameters are listed for your convenience.

Table 3-2 *Default Calling Parameters*

Note: Your applications will not run if you do not make these changes. References to deleted routines will cause compilation errors.

```
Bool defaults_exists(name, class)
Bool defaults_get_boolean(
                   name, class, default_bool)
char defaults_get_character(
                   name, class,
                   default_character)
char * defaults_get_string(
                   name, class, default_string)
int defaults_get_enum(name, class, pairs)
int defaults_get_integer(
                   name, class, default_inte-
ger)
int defaults_get_integer_check(
                   name, class, default_int,
                   minimum, maximum)
int defaults_lookup(name, pairs)
void defaults_init_db()
void defaults_load_db(filename)
void defaults_set_character(resource, value)
void defaults_set_integer(resource, value)
void defaults_set_string(resource, value)
void defaults_store_db(filename)
```

`defaults_lookup()` has not changed since SunView. All the others have new parameters. Routines not listed in table have been deleted.

Initialization

To make sure objects have the correct parent, XView applications need to know what server and screen on which to connect. Use `xv_init()` to provide this information. The code below returns the server object that was created, or `XV_ERROR` if a failure.

```
server = (Xv_server)xv_init(XV_INIT_ARGS, argc,
   argv, NULL);
```

To make conversion easier, `xv_init()` is called automatically during the first run-time call to `xv_create()` or `pf_open()`. It may fail, however, if the first call to `xv_create()` does not include `XV_ARGC_PTR_ARGV`, `XV_INIT_ARGS`, `FRAME_ARGC_PTR_ARGV`, or `FRAME_ARGS`, or if that first call creates some other object during processing. For instance, in the following code fragment, missing arguments cause the program to fail.

```
 xv_create(NULL, FRAME,
FRAME_ICON,xv_create(ICON,ICON_IMAGE,my_image,0),
FRAME_ARGC_PTR_ARGV,  &argc, argv, . . . 0);
```

For more detailed information on initializing XView, refer to the XView Programming Manual.

`DEFINE` Macros

The `cursor` and `icon` objects are no longer public `structs`, so the following `#DEFINE` statements are removed. Failure to make these changes could cause compile-time errors.

Table 3-3. *Removed #DEFINE Items*

#DEFINE	***Comments***
`DEFINE_CURSOR`	Use `xv_create(0, CURSOR,...)`
`DEFINE_CURSOR_FROM_IMAGE`	
`DEFINE_ICON_FROM_IMAGE`	Use `xv_create (0,ICON,...)`

The following definitions have been removed from `<sunwindow/rect.h>`:

`min`	Use `MIN` in `base.h`
`max`	Use `MAX` in `base.h`
`bool`	
`TRUE`	Moved to `base.h`
`FALSE`	

Input Events

A number of input events are no longer generated by the system. These are shown in the table below. Failure to make these changes could cause compile-time errors. In addition, two other changes have occured. These are as follows:

1) `event_is_meta`, `META_FIRST` and `META_LAST` have been deleted. Holding Meta down while hitting a key will generate an event in the ASCII range, but with the meta shift bit set in the shiftmask.

2) `event_is_iso`, `ISO_FIRST` and `ISO_LAST` have been defined. `ISO_FIRST` is 0, and `ISO_LAST` is 255. This adds support for the 8-bit ISO Latin 1 character set.

Table 3-4 *Obsolete Input Events*

Event Name	***Comments***
`KBD_REQUEST`	Click-to-type input focus handled by an external window manager; this event is not generated.
`LOC_STILL`	This cannot be generated (easily) on top of the X11/NeWS server.
`LOC_TRAJECTORY`	The X11/NeWS server automatically compresses events; compression can only be disabled for *all* windows (which is very expensive for performance).
`LOC_RGNENTER`	No `pixwin` regions in XView, everything is a window.
`LOC_RGNEXIT`	Same as `LOC_RGNENTER`.
`WIN_STOP`	This is now an ordinary function key event. In SunView, it is generated asynchronously via a signal which isn't supported by the X11/NeWS server. (Signals are valid in a kernel-based window system but have no place in a networked window system, so there is no more `SIGURG`, either.) In XView, since `WIN_STOP` is equated to `KEY_LEFT(1)`, `WIN_STOP` events are generated by the `L1` (`Stop`) function key.

Old Input Handling

Use `WIN_CONSUME_EVENT and WIN_CONSUME_EVENTS` instead of `win_setinputmask()` to manipulate the input mask. For example, replace the following:

```
(void)input_imnull(&im);
im.im_flags |= IM_ASCII;
(void)win_setinputmask(msgsw->msg_windowfd, &im,
   (struct inputmask *)0, WIN_NullLink);
```

with

```
xv_set(mywindow, WIN_CONSUME_EVENTS,
   WIN_ASCII_EVENTS)
```

Mouse Events

Due to limitations in the X protocol, an application cannot register interest on a specific mouse button. An application can only register interest on all mouse buttons, or none at all. Registering interest on one mouse button, also registers interest on all available mouse buttons. In terms of XView, if you consume `MS_LEFT`, then you are also consuming `MS_MIDDLE` and `MS_RIGHT`. To work around this, your event proc should disregard button events that your application is not interested in.

On a related note, to ignore mouse button events (unregister interest) use: `WIN_IGNORE_EVENTS`, and `WIN_MOUSE_BUTTONS` in an `xv_set` call. Do not use: `WIN_IGNORE_EVENTS`, `MS_LEFT`, `MS_MIDDLE`, `MS_RIGHT`.

Event Timestamps

The time stamp of events returned by `event_time()` is the same `timeval` structure as before. However, in X11, event timestamp resolution is in milliseconds. Also, the timestamp of the event is measured as the time since the server started, not the time-of-day. If you are using event timestamps as the time-of-day in SunView, you can simulate this approximately in XView with the following fix: at initialization, compare the timestamp of the first X11 event with `localtime()`, then offset subsequent event times accordingly

Input

XView has only one input reading function per server, not one per window FD.

Structures

The following `structs` either have been removed, are no longer public information, or are changed. Programs that refer to these structures must remove the references and use appropriate `get/set` calls.

struct pixwin

Defined in `<sunwindow/pixwin.h>`. A `pixwin` is now an opaque object. The type is only provided for compatibility. All the `pixwin` drawing routines can still be called, but the fields of the structure are no longer available.

struct singlecolor

With the advent of SunOS 4.1, the definition of `struct singlecolor` has been dropped from `/usr/include/pixrect/pixrect.h`, and `CMS_NAMESIZE` has been defined. These changes require the following actions:

- Modify your XView sources to use `struct xv_singlecolor` instead of `struct singlecolor`.
- Do not include `pixrect_hs.h`. If necessary, include only those files listed in `pixrect_hs.h` that do not include sunwindow/cms.h, namely the framebuffer include files (e.g., cg2var.h).

These changes will not affect your ability to compile and run on SunOS 4.0 machines.

struct toolsw

Defined in `<suntool/tool_struct.h>`. This is also pre-SunView.

struct cursor

This `struct` is defined in `<sunwindow/win_cursor.h>`. Clients should use cursor attributes to manipulate cursors instead.

struct icon

Defined in `<suntool/icon.h>`. Clients should use icon attributes instead. Icons are separate windows in XView, not different states of frames.

struct screen

Defined in `<sunwindow/win_screen.h>`. Replaced by the `SERVER` and `SCREEN` objects.

struct fullscreen

Defined in `<suntool/fullscreen.h>`. This structure is changed, not removed.

In XView, you take over the root window, and can draw on it in screen coordinates, but you still receive events in the coordinates of the window taking over the screen. There are special routines to set the cursor and input mask: `fullscreen_setcursor()` and `fullscreen_setinputmask()`. The fields in the `fullscreen` structure that are no longer supported are:

Table 3-5 *Unsupported Struct Fullscreen Fields*

Struct Fullscreen Fields	***Comments***
`fs_pixwin`	Instead you can draw directly on screen using the window handle `fs_rootwindow`.

`fs_cachedcursor`
`fs_cachedim`
`fs_cachedinputnext`
	No need to restore state since fullscreen no longer uses your window.
	window.`fs_cachedkbdim`

struct inputmask

Defined in `<sunwindow/win_input.h>`. This structure is not removed but changed. The fields no longer supported are shown below.

Table 3-6 *Unsupported struct inputmask Fields*

Struct inputmask	***Comments***

`im_inputcode`
`im_shifts`
`im_shiftcodes`
	Access the inputmask through `WIN_CONSUME_EVENT`

Command Line Processing

The following routines, documented only in the *SunWindows Reference Manual*, have been removed.

Table 3-7 *Removed Command Line Functions*

Command Line Function	***Comments***
`tool_parse_all()`	Use `XV_ARGC_PTR_ARGV`
`tool_parse_one()`	Use `XV_ARGS`
`tool_usage()`	Use `xv_usage()`

Full-Screen Mode

In XView, clients must draw in screen coordinates, rather than in the coordinates of the window that was used to go full-screen. Events are still reported in the coordinates of the window that becomes full-screen. They must also use the new functions `fullscreen_set_cursor()` and `fullscreen_set_inputmask()`.

3.3 Alert Package

The SunView Alert package no longer exists. The XView Notice package replaces it completely. You may find it useful to `#define` the following constants:

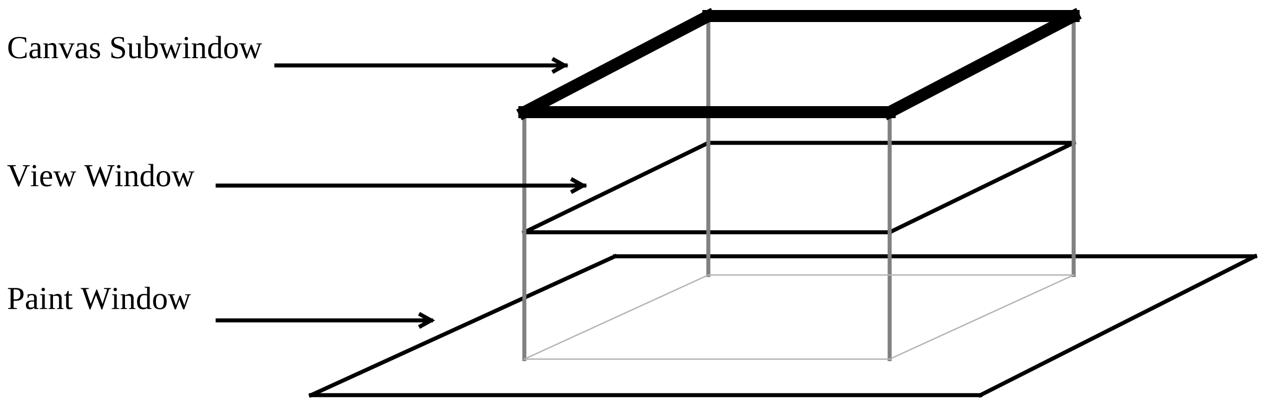
```
#define ALERT_YES           1
#define ALERT_NO            0
#define ALERT_FAILED       -1
#define ALERT_TRIGGERED    -2
#define alert_prompt()     notice_prompt()
```

3.4 Canvases

Since canvases are no longer retained, as they were in SunView, you now have to handle repainting yourself. This makes the canvas paint window more important: you interpose on it to get events, and it is where you must do all paint operations. It is easiest to do them in the canvas paint procedure, which is called when a repaint event is delivered to the canvas paint window. Thus, painting to a canvas paint window before the window is actually mapped to the screen is a waste of time.

As shown in the following diagram, canvases are implemented as three separate windows, the *canvas subwindow*, the *view window* and the *paint window*.

The Canvas Model



The paint window is a child of the view window, which is in turn, a child of the canvas subwindow. The view window clips the paint window to that area of the canvas subwindow unobscured by the scrollbars. The paint window may be much larger than either the view window or the canvas subwindow, in which case only a small portion of it will be visible at a given time.

All canvas paint windows have a bit gravity associated with them. The bit gravity of a window tells the server which way to move the bits in a window when it is resized. For example, if you have `NorthGravity` set and resize a window larger (from the bottom), the bits will stay where they are and you will receive a `WIN_REPAINT` for the newly exposed area under your image. If you have `SouthGravity` set and perform the same resize operation, the image in the canvas will be pulled south with the resize, and your canvas will receive damage on the newly exposed area at the top of the window (even though you resized from the bottom). Bit gravity should be used for fixed images. If your canvas contains an image that is dependent on the size of the canvas paint window (for example, if it renders small images in columns), then you should not use bit gravity. If you set bit gravity to `ForgetGravity`, then whenever your canvas is resized, the entire canvas, not just the newly exposed portions, will become damaged, and will need to be repainted.

You can set bit gravity with `xv_set` or `xv_create`, using `WIN_BIT_GRAVITY`. Valid values are:

```
NorthWestGravity (default)
NorthGravity         NorthEastGravity
WestGravity          EastGravity
CenterGravity        SouthWestGravity
SouthGravity         SoutEastGravity
StaticGravity        ForgetGravity
```

To force bit gravity to be `ForgetGravity` when an image depends on the size of the canvas (that is, for an image that changes as the canvas size changes), set `CANVAS_FIXED_IMAGE` to be `FALSE`. This is basically equivalent to using `WIN_BIT_GRAVITY`.

When porting canvas properties to XView, be aware of the following changes:

1) Replace `LOC_RGNENTER` with `LOC_WINENTER.` Scrollbars are implemented as separate windows.
2) In XView, the part of the canvas you draw on is a separate window called the *canvas paint window*. It is not part of the canvas subwindow itself, so set the cursor on the canvas paint window, and not on the canvas. The canvas paint window is returned by function `canvas_paint_window()`, and the old `canvas_pixwin()` function.
3) Set `WIN_CONSUME_EVENTS` on the canvas paint window to read input events from a canvas.

Note: The server usually gives you a retained canvas (backing store) when you ask for it; however, the canvas will become unretained when the server runs low on memory.

4) `CANVAS_RETAINED` is only a hint to the server. According to the *X11 Specification*, your application must be prepared to repaint itself at any time, even though it may have asked for a retained canvas. A canvas *may* change from retained to unretained after being created.
5) SunView's `pw_*` calls provided automatic clipping. However, if your image depends on the size of your canvas, or if your paint window is larger than your view window, you will probably want to use `CANVAS_NO_CLIPPING` along with `CANVAS_FIXED_IMAGE` set to `FALSE`.
6) Replace `canvas_window_event()` with `event_window()`.

Run-Time Incompatibilities

The following sections describe canvas conversion issues that may cause run-time errors if not implemented.

Bit Gravity

XView now takes advantage of bit gravity on resize and repaint events. This helps to improve performance by eliminating unnecessary repaints. With bit gravity, when a window is resized, only the changed area (the difference between the old and new regions) is repainted. The default for bit gravity is NorthWestGravity. See WIN_BIT_GRAVITY and WIN_WINDOW_GRAVITY in the *XView Programmers Manual.*

Canvas Subwindow Events

The canvas subwindow event function now only gets events for the canvas subwindow itself, so interposers on the canvas subwindow event function will not see events in the canvas *paint* window. Most clients will need to interpose on the paint window, which they can retrieve with the CANVAS_NTH_PAINT_WINDOW attribute.

3.5 Cursors

This section describes changes to the cursor package.

1) The X11/NeWS server supports a mask type cursor, so it cannot support all cursor RasterOp logical operations. Further, in SunView, the cursor structure was accessible to programmers. Since this is not the case in XView, you must change static cursor initialization, as shown below, and remove all references that begin #DEFINE_CURSOR_FROM_IMAGE.

```
struct cursor    hourglass_cursor =
    {8,8,PIX_SRC | PIX_DST, &hourglass_cursor_pr }
```

2) To manage the cursor in XView, call the code shown below in main().

```
Xv_Cursor hourglass_cursor;
hourglass_cursor = xv_create(NULL, CURSOR,
    CURSOR_IMAGE,    hourglass_cursor_pr,
    CURSOR_XHOT,     8,
    CURSOR_YHOT,     8,
```

3) To get a cursor, replace

```
win_getcursor(fd, &oldcursor)
```

with

```
oldcursor = xv_get(window, WIN_CURSOR);
```

4) To set a cursor, replace calls such as

```
win_setcursor(fd, &mycursor)
```

with

```
xv_set(window,
       WIN_CURSOR, mycursor,
       0);
```

5) Remove, or use `#ifdef SV1`, on all crosshair attributes.
6) In SunView, you could create a cursor, hand it to a pointer to `pixrect`, change the bits in that `pixrect`, then use `WIN_CURSOR` to set the new cursor image. In XView, you must explicitly set the cursor to the new image using the existing `CURSOR_IMAGE` attribute.

Compile-Time Incompatibilities

All crosshair attributes from the cursor package in `<sunwindow/win_cursor.h>` are unsupported.

Removed Cursor Attributes

Cursor Attribute	***Comments***
`CURSOR_SHOW_CROSSHAIRS`	None of these
`CURSOR_SHOW_HORIZ_HAIR`	attributes exists
`CURSOR_SHOW_VERT_HAIR`	in XView
`CURSOR_CROSSHAIR_BORDER_GRAVITY`	
`CURSOR_HORIZ_HAIR_BORDER_GRAVITY`	
`CURSOR_VERT_HAIR_BORDER_GRAVITY`	
`CURSOR_CROSSHAIR_COLOR`	
`CURSOR_CROSSHAIR_OP`	
`CURSOR_CROSSHAIR_THICKNESS`	
`CURSOR_CROSSHAIR_LENGTH`	
`CURSOR_CROSSHAIR_GAP`	
`CURSOR_HORIZ_HAIR_COLOR`	
`CURSOR_HORIZ_HAIR_OP`	
`CURSOR_HORIZ_HAIR_THICKNESS`	
`CURSOR_HORIZ_HAIR_LENGTH`	
`CURSOR_HORIZ_HAIR_GAP`	
`CURSOR_VERT_HAIR_COLOR`	
`CURSOR_VERT_HAIR_OP`	
`CURSOR_VERT_HAIR_THICKNESS`	
`CURSOR_VERT_HAIR_LENGTH`	
`CURSOR_VERT_HAIR_GAP`	

Cursor Run-Time Incompatibilities

The sections below describe cursor properties that must be implemented in order to avoid run-time incompatibilities.

Hot Spots

In XView, if the hot spot coordinates are greater than the cursor's pixrect, the pixrect's size is increased to include the hot spot; however, this does not handle cursors with negative hot spot coordinates.

Unsupported Cursor Raster Operations

X11/NeWS does not directly support cursor raster operations. All but seven of these have been implemented with a cursor source and mask. The unimplemented cursor raster operations are shown below:

Table 3-9 *Unimplemented Cursor Raster Operations*

Unimplemented Cursor RasterOps
`PIX_SRC XOR PIX_DST`
`PIX_SRC AND PIX_NOT(PIX_DST)`
`PIX_NOT(PIX_SRC) AND PIX_NOT(PIX_DST)`
`PIX_NOT(PIX_SRC) XOR PIX_DST`
`PIX_SRC OR PIX_NOT(PIX_DST)`
`PIX_NOT(PIX_SRC) OR PIX_NOT(PIX_DST)`
`PIX_NOT(PIX_DST)`

Changing Cursors

In SunView, each window has its own unique cursor. In XView, cursors are shared. Thus, if you change a window's cursor but do not use `cursor_copy()` to create a copy of a cursor, the cursor may change in other windows as well.

3.6 Fonts

Fonts are handled differently in XView than in SunView. In Sunview when you wanted to do a graphics operations that involved fonts, you simply passed a pixfont handle in the argument list to whatever graphics function you were calling. In X you have to create a graphics context prior to calling a graphics function. One of the attributes of the graphics context is a font. Then, when a function is called, you simply pass the graphics context with the pre-figured font.

Use the following font guidelines when converting SunView to XView.

1) Although you can access a copy of your font in `Pixfont` format, there are some things you cannot do with it:
   - You cannot reference a glyph's `pixrect`, change it, and then expect to see the altered character on the screen.
2) XView has no equivalent for SunView's `pw_pfsysopen()` routine, which opens the default font, because in XView, your program may be talking to multiple remote servers, each with its

own default font. To get the same effect in XView, use either of the following code constructs:

```
default_font = xv_create(frame, FONT, 0);
    /* or */
default_font = xv_find(frame, FONT, 0);
    /* Where frame can be any window which tells
     * XView which server you're interested in. */
```

To get the `default_font` on the default server do either of the following:

```
default_font = xv_create(0, FONT, 0)
    /* or */
default_font = xv_find(0, FONT, 0)
```

Note that `xv_create` creates an XView object, and `xv_find` looks for an object which has the given specifications. If one does not exist, it is created.

3) Accessing different fonts in XView is different than in SunView. To access a font in SunView you would use the following:

```
mypf = (Pixfont *)pf_open("/usr/lib/fonts/
      fixedwidthfonts/mumble.b.14")
```

In XView you would use the object-oriented `XV_find` interface with the `FONT_NAME` attribute:

```
xvfont = (Xv_Font)xv_find(frame, FONT,
   FONT_NAME, "fontname",
   0);
```

4) Fonts are shared on the server, so if the font was already requested by another client (process), then the server doesn't need to allocate the font again (unlike under SunView). The fonts you use on the server must already have been compiled. When the X server is started, a list of font paths can be specified using the `xinit` command.

5) Fonts are read-only, and cannot be changed. Fonts are property of the server. Clients can only get a data-structure which describes the font, never the bits of the images.
6) To query the server and print out all the font names, use the X program `xlsfonts`. The program `xfd` previews fonts (see the XX reference manual pages.
7) `xv_create(..., FONT, ...)` and `xv_find(..., FONT, ...)` do not return a pointer to a Pixfont. They now return a pointer to a standard XView object (`Xv_font`). This means that you can no longer treat the returned pointer as a (Pixfont *) and do things such as the following:

```
Pixfont *pf;
pf = xv_create(..., FONT, ...);
width = pf->pf_defaultsize.x;
```

Included below is a guide to converting code such as the above.
8) Attributes to get font information. There are two attributes which can be used to access structures that contain information on the particular font. `xv_get(font, FONT_INFO)` returns a pointer to an `XFONTStruct` (an Xdata structure). `xv_get(font, FONT_PIXFONT)` returns a pointer to a pixfont.
9) Convert code that uses Pixfonts. In converting code that uses pixfonts, one should avoid the use of the attribute `FONT_PIXFONT` as much as possible (it is referred to only twice below). This is important because in XView `XFontStruct` (the structure obtained via `FONT_INFO`) is automatically allocated when a font is created. When the attribute `FONT_PIXFONT` is used, about 5Kbytes of memory is allocated (one done once) for the particluar font. An example of converting code that uses pixfonts is shown below:

SunView Method:

```
offset=pf->pf_char[i].pc_home.x;
```

XView Method:

```
x_font_info=(XFontStruct *)xv_get(pf, FONT_IN-
FO);
offset=x_font_info->per_char
```

A conversion table for converting code that uses pixfonts is shown in Table 3-10. The variable declarations and include files that pertain to the table is show first:

```
#include   <X11/Xlib.h>
#include   <pixrest/pixfont.h>
#include   (xview/font.h>

Pixfont *pf;
Pixfont *pixfont;
XFontStruct*x_font_info;
```

Table 3-10 *Converting Pixfonts to XView*

SunView Method	***XView Method***
pf->pf_defaultsize.x	xv_get(pf, FONT_DEFAULT_CHAR_WIDTH)
pf->pf_defaultsize.y	xv_get(pf, FONT_DEFAULT_CHAR_HEIGHT)
pf->pf_char[i].pc_home.x	x_font_info = (XFontStruct *)xv_get(pf, FONT_INFO); x_font_info->per_char[i-x_font_info->min_char_or_byte2].lbearing
pf->pf_char[i].pc_home.y	x_font_info = (XFontStruct *)xv_get(pf, FONT_INFO); -(x_font_info->ascent)
pf->pf_char[i].pc_adv.x	x_font_info = (XFontStruct *)xv_get(pf, FONT_INFO); x_font_info->per_char[i - x_font_info->min_char_or_byte2].width
pf->pf_char[i].pc_adv.y	0
pf->pf_char[i].pc_pr.pr_height pf->pf_char[i].pc_pr.pr_size.y	x_font_info = (XFontStruct *)xv_get(pf, FONT_INFO); x_font_info->ascent + x_font_info->descent
pf->pf_char[i].pc_pr.pr_width pf->pf_char[i].pc_pr.pr_size.x	x_font_info = (XFontStruct *)xv_get(pf, FONT_INFO); x_font_info->per_char[i-x_font_info->min_char_or_byte2].width
pf->pf_char[i]	pixfont = (Pixfont *)xv_get(pf, FONT_PIXFONT); pixfont->pf_char[i]
pf->pf_char[i].pc_pr	pixfont = (Pixfont *)xv_get(pf, FONT_PIXFONT); pixfont->pf_char[i].pc_pr

Font Compile-Time Incompatibilities

Be sure to use the font package (e.g., use `FONT_STRING_DIMS` instead of `pw_text`). Here is a list of changed functions:

Table 3-11 *Changed pf_* Functions*

Old Function Name	*New Function Name*
`pf_sys`	`xv_pf_sys`
`pf_open()`	`xv_pf_open()`
`pf_close()`	`xv_pf_close()` `==>actually defunct`
`pf_text()`	`xv_pf_text()`
`pf_textbatch()`	`xv_pf_textbatch()`
`pf_textbound()`	`xv_pf_textbound()`
`pf_textwidth()`	`xv_pf_textwidth()`
`pf_ttext()`	`xv_pf_ttext()`
`pf_default()`	`xv_pf_default()`

Font Run-Time Incompatibilities

`pf_default()` no longer examines the environment variable `DEFAULT_FONT`, but takes the default font from the defaults database. Note that `pf_*` is now `xv_pf_*` and must be externed in the client code (cannot include a header file).

3.7 Frames

The frame menu is handled by an external window manager in XView. This cannot be modified by the application.

When converting code involving frames:

1) See the cautions about using `FRAME_ARGC_PTR_ARGV` in the subsection above on *Initialization*.
2) Replace `FRAME_NTH_WINDOW` by iteration over the existing attributes `FRAME_NTH_SUBFRAME` and `FRAME_NTH_SUBWINDOW`.
3) To place frame in a particular location relative to another frame, replace the use of `WIN_X` and `WIN_Y` with the `frame_set_rect()` function::

```
frame_set_rect(frame, rect)
   Frame        frame;
   Rect         *rect;
```

To retrieve the current position of a frame relative to the root window use:

```
frame_get_rect(frame, rect)
   Frame        frame;
   Rect         *rect;
```

Note that this code works only for frames).

4) The frame window still exists in XView, but the responsibility for drawing the border around the tool, displaying the label in the frame header, and handling the frame menu lies with an external window manager. The frame is still the container for the subwindows, but it doesn't *poke out* around the subwindows. Consequently:

- Do not attempt to access or change the frame's menu, since this will be under the control of an external window manager.
- The geometry of subwindow layout and frame positioning is subtly different, since the frame window has a zero-width border in XView.

Compile-Time Incompatibilities

The frame attribute `FRAME_NTH_WINDOW` is no longer supported. It can be replaced by iteration with the attributes `FRAME_NTH_SUBFRAME` and `FRAME_NTH_SUBWINDOW`.

Run-Time Incompatibilities

In the frame command line arguments, only the very first object created will process the `FRAME_ARGS` or `FRAME_ARGC_PTR_ARGV` attributes. The command line is parsed into entries in the defaults database, which are then queried by the appropriate packages, as in the following:

```
% -Wh → /XView/Cmdline/Height
```

Most of the command line arguments refer to frame defaults. The frame package applies the command line defaults only to the first base frame that is created. For example, `-Wp 100 150` positions the first base frame that is created at `(100,150)`. The window manager computes a default position for each subsequently created base frame.

Iconic Frame Events

The frame no longer gets events when iconic, since a frame's icon has its own window.

3.8 Icons

If your application has an odd-sized icon, you were probably manipulating the icon's internal data structures. This is not allowed in XView. Instead, you work with icons through an icon pointer, which you get at create time. From that pointer, you can manipulate the icon's remote image.

1) Replace

```
DEFINE_ICON_FROM_IMAGE(my_icon, my_ic_image);
```

with the following code:

```
mpr_static(my_icon, ICON_DEFAULT_WIDTH,
       ICON_DEFAULT_HEIGHT, 1, my_ic_image);
Icon  my_icon;
/*  Later on, in main() */
my_icon = xv_create(NULL, ICON, ICON_IMAGE,
        &my_icon,0);
```

2) Use attributes where you manipulated fields of the icon structure in SunView.

Changing a Frames Icon.

The following code sequence is for changing the icon image and label.

```
icon = xv_get(frame, FRAME_ICON);
xv_set(icon,
   ICON_IMAGE, &d_icon_pr,
   ICON_LABEL, "foo",
   0);
xv_set(frame,
   FRAME_ICON, icon,
   0);
```

Icon Run-Time Incompatibilities

In XView, icons are separate windows. If a frame is iconic and you change its icon, its image on-screen will change immediately; in SunView the image would not change until you called `window_set(frame, FRAME_ICON, icon, 0);`

3.9 Menus

Remove code that tries to change a frame's menu, since the frame's menu is under the control of the external window manager in XView.

Also, you can get notification that any menu in a menu group is done by attaching `MENU_DONE_PROC` to each menu. However, you will get better results with `menu_item` action procedures.

Compile-Time Incompatibilities

Stacking menus and prompt boxes are no longer supported. Stacking menus are replaced by the Walking Menu Package. Prompt boxes are replaced by the Notices package. The following `structs` and functions are no longer supported:

Table 3-12 *Unsupported Menu Structures and Functions*

Struct/Functions	*Comments*
`struct menu`	Use `xv_create(0,MENU,...)`and `xv_set/get()`
`struct menu_item`	Use `xv_create(0,MENU_ITEM,...)`
`menu_display()`	Use `menu_show()`
`struct prompt`	Use `notice_prompt()` with its
`menu_prompt()`	various attributes.

Clicking `Select` (Left button) on a menu button to select the default menu item will not cause the attached menu's `done` procedure to be called; no menu is shown. For better results, use `action procs` on each item. Use the following guidelines:

- Allow the menu to be pinned automatically (via `MENU_GEN_PIN_WINDOW`);
- Use action procs. Action procs involve less code; you do not have to do `xv_get` to find the selected item; and they work no matter how the menu item was selected.

3.10 Panels

The OPEN LOOK UI specifies different panel images for buttons, choice items, and so on, that differ from SunView. In general, all panel images must now be `SERVER_IMAGE` objects, rather than memory pixrects. These attributes include: `PANEL_CHOICE_IMAGE`, `PANEL_CHOICE_IMAGES`, and `PANEL_LABEL_IMAGE`. The difference between server images and memory pixrects are that server images are the XView object abstraction of an Xlib Pixmap. They are bitmaps that reside on the X11 server. A memory pixrect is a bitmap that is stored as data on the client side. If you have been laying out panels explicitly in pixel coordinates, you will now have to specify panel item locations. Also, the panel package has been integrated with the walking menus package, and most `PANEL_MENU_*` attributes are replaced by menu attributes.

Panel Compile-Time Incompatibilities

The panel menu package is not supported in XView. Clients should use the menu package to replace the semantics of the following `PANEL_MENU` attributes. Note that if your SunView code used `PANEL_LABEL_IMAGE` in conjunction with panel_button_image(), use PANEL_LABEL_STRING. For example, the following SunView code:

```
panel_button_image(panel, "Button", G, my_pf_font)
```

would be replaced by the following XView code.

```
PANEL_LABEL_STRING, "Button",
```

Table 3-13 *Removed PANEL_MENU Attributes*

Panel Attribute	***Comments***
PANEL_MENU_CHOICE_FONTS	Use XV_FONT to set the font of each menu item.
PANEL_MENU_CHOICE_IMAGES	Use MENU_IMAGE
PANEL_MENU_CHOICE_STRINGS	Use MENU_STRING
PANEL_MENU_CHOICE_VALUES	Use MENU_VALUE
PANEL_MENU_TITLE_FONT	Unsupported: menu titles can't be given a different font.
PANEL_MENU_TITLE_IMAGE	Use MENU_TITLE_IMAGE
PANEL_MENU_TITLE_STRING	Use MENU_TITLE_STRING

In addition, the following pre-SunView attributes have also been removed.

Table 3-14 *Removed Pre-SunView Panel Attributes*

Panel Attribute	***Comments***
PANEL_WIDTH	Use XV_WIDTH
PANEL_HEIGHT	Use XV_HEIGHT
PANEL_VERTICAL_SCROLLBAR	Use WIN_VERTICAL_SCROLLBAR
PANEL_HORIZONTAL_SCROLLBAR	Use WIN_HORIZONTAL_SCROLLBAR
PANEL_PIXIN	Use WIN_PIXWIN
PANEL_CU	Use the new xv_row() and xv_col() functions instead
PANEL_FIT_HEIGHT	Use WIN_FIT_HEIGHT

Removed Panel Functions

The following routines have been removed. They are only documented in the *SunView 1 System Programmer's Guide* or the (obsolete) *SunWindows Reference Manual*.

Table 3-15 *Removed Panel Functions*

Panel Function	***XView Replacements***
`panel_make_list()`	Use `attr_create_list()`
`panel_fit_height()`	Use `window_fit_height()`
`panel_fit_width()`	Use `window_fit_width()`
`panel_window_event()`	Use `event_window()`

Run-Time Incompatibilities

The default images supplied for graphic panel components (buttons, choice items, and so on) are different in XView. Consequently, SunView applications which explicitly lay out their panel items in pixel coordinates will end up with overlapping or otherwise jumbled panel items until the program is changed to specify new pixel coordinates or, better yet, to use row/column positioning.

Pre-SunView Panel Interface

This section describes the procedures for converting pre-SunView panel code to XView.

1) To create a panel, replace
   `panel = panel_create(tool, 0);`
   with `panel = xv_create(frame, PANEL, 0);`
2) To size the height of a panel to fit its items, replace
   `(void)panel_set(panel, PANEL_HEIGHT, PANEL_FIT_ITEMS, 0);` and
   `panel_fit_height(panel)` with
   `window_fit_height(panel);`
   The same applies to panel width. You can call `window_fit(panel)` to size a panel to fit its items in both dimensions.
3) Replace `PANEL_HEIGHT` with `WIN_HEIGHT` and replace calls to `panel_get()`with `xv_get()`.
4) `PANEL_CU()` is gone. If you are not doing a full XView conversion and are using `window_create()` and `panel_create_item()`, you can use `ATTR_ROW()`and `ATTR_COL()`instead. If you are doing a full conversion to `xv_create()`, use the new `xv_row()` or `xv_rows()` and

`xv_col()` or `xv_cols()` functions, depending on whether you used `PANEL_CU()` to set rows or columns.

5) Replace `win_get_fd(panel)` with `xv_get(panel, WIN_FD);` If you are doing a complete conversion, just use the window itself in any call. (`WIN_FD` is a no-op and is `#defined` to `XV_SELF.`)

6) SunView's `PANEL_CYCLE` item is now equivalent to an XView `PANEL_TOGGLE_ITEM`. (The OPEN LOOK UI description is `Panel Choice Stack` item.) Functionally, the difference is that in SunView, pressing the left mouse button on the appropriate glyph toggles the options; in XView, the options are selected from a menu.

3.11 Pixwins

This section briefly describes the procedure for converting pixwins. Detailed Pixwin to Xlib conversion instructions are provided in Appendix E.

The `pixwin struct` is not a part of XView. For compatibility, you can still ask for a window's pixwin and use it in as many routines as before, but the pixwin returned is simply the window handle, which is a strictly opaque object whose fields you cannot access. However, in XView, you can draw on a window directly with its window handle. The `pixwin` returned by `canvas_pixwin()` and `WIN_PIXWIN` is really just the window handle, and most unsupported `pixwin` operations are just performance boosters or setups under SunView that no longer apply under X11.

XView does not support pixwin regions. Instead, you create multiple windows. However, if you used `pw_traprop()`, which is no longer supported, then you will have to rethink your graphics output routines. One alternative is to use `pr_traprop`() to draw into a memory `pixrect`, then use that as the source in an `xv_rop()` operation.

Some other unsupported calls include routines for locking, batching, and double-buffering. Functionality at this level is the responsibility of the window server, not the application.

Compile-Time Incompatibilities

The following section describes Pixwin properties that must be changed in order to avoid compile-time problems.

Type Definitions

These `<sunwindow/pw_dblbuf.h>typedefs` are not supported.

Table 3-16 *Unsupported Type Definitions*

Typedef	***Comments***
`Pw_attribute_value` `Pw_dbl_attribute`	Basically, X11/NeWS does not support double-buffering.

Pixwin Functions and Procedures

The following routines shown in Table 3-17 no longer exist; all references must be removed from XView programs. Note, however, that some new window attributes have been added to provide similar functionality. Note also that clients cannot access to the fields in a `pixwin struct`. Because the damage list field of the `pixwin struct` is also no longer available, `pw_read()` and `pw_copy()` do not provide a fix-up list. However, a new function, `win_get_damage()`, can help clients to determine the damage list when processing a `WIN_REPAINT` event.

`Pixwin` regions are not supported (use real windows instead), and double-buffering is also not supported.

Table 3-17 *Removed Pixwin Functions*

Removed Pixwin Function	***Comments***
`pw_open/close()`	No need to open/close a window. Since the `pixwin` is the window, it goes away when the window goes away.
`pw_destroy()`	
`pw_region()`	Use real windows instead.
`pw_restrictclipping()`	Use `win_set_clip` instead.
`pw_exposed()`	Windows are clipped for you by the X11/NeWS server, so justset clipping to the whole window: `win_set_clip(-window`, `RECTLIST_NULL)`
`pw_damaged()`	Use `win_get_damage`.
`pw_donedamaged()`	Use `win_get_damage`.
`pw_repairretained()`	
`pw_set/get_x_offset()`	Offsetting is not supported.
`pw_set/get_y_offset()`	
`pw_set/get_xy_offset()`	
`pw_preparesurface()`	unnecessary
`pw_set/get_region_rect()`	Most regions in SunView1 have become full-fledged windows in XView, so you can often use `WIN_RECT` instead.
`pw_dbl_get()`	Double-buffering is not supported.
`pw_dbl_access()`	Double-buffering is not supported.
`pw_dbl_release()`	Double-buffering is not supported.
`pw_dbl_flip()`	Double-buffering is not supported.
`pw_dbl_set()`	Double-buffering is not supported.
`pw_traprop()`	Not implemented.
`pw_use_fast_monochrome()`	No substitute. Set the new `WIN_VISUAL` attribute to `WIN_MONOCHROME.`
`pr_load()`	To create a pixmap with depth from a specified X bitmap file, use `XCreatePixmapFromBitmapData`
`pr_dump()`	In Xlib use `XWriteBitMapFile`.

Double-Buffering Function and Procedures

Since X11/NeWS doesn't support double-buffering, the following double-buffering attributes have been removed:

Table 3-18 *Removed Double-Buffering Attributes*

Double-Buffering Attribute	***Comments***
`PW_DBL_ERROR`	All gone
`PW_DBL_FORE`	
`PW_DBL_BACK`	
`PW_DBL_BOTH`	

Pixwin Run-Time Incompatibilities

The following routines are replaced or become no-ops. Programs that refer to them will compile but their semantics are no longer the same. The exact functionality of `pixwin` batching, as with `pw_batch()`, is not provided; however the X11/NeWS server batches the drawing operations automatically until the next event is read or until a request which requires a reply is made to the server. XView flushes these batched operations when `pw_batch_off()` is invoked, thus forcing the drawing operations to take place.

Table 3-19 *Changed Pixwin Functions*

Pixwin Function	***Comments***
`pw_batch()`	Batching is not
`pw_batch_on()`	supported by X11/NeWS.
`pw_reset()`	
`pw_preparesurface_full()`	There is no need to prepare a
`pw_preparesurface()`	surface, since all drawing
	takes place in a real window.

pw_polygon2()

The `pw_polygon2()` routine is implemented for its simple cases, but some of its more complicated semantics are not implemented. Source for filling the polygon can be either a `pixrect` or a `server_image`.

Foreground and Background Pixel Values

When converting your Pixwin/Pixrect graphics operations to Xlib, do not make any assumptions about the foreground and background pixel values. In SunView they were logical ’0’ and ’1’. In the X Window System, this is not the case and many different X servers use different values that would cause serious display problem if this principle were violated. For your convenience, Xlib provides macros for getting at the logical "black" and "white" pixel values. Thes are as follows:

```
foreground = BlackPixel(dpy, scr);
background = WhitePixel(dpy, scr);
```

3.12 Scrollbars

OPEN LOOK UI scrollbars are different from the SunView scrollbars. Most applications are not affected by this change in the user interface, but applications that use advanced features of the scrollbar programmatic interface may require some changes. To use scrollbars properly, convert to the new calls and attributes. Use `SCROLLBAR_DEFAULT` instead of `SCROLLBAR` to get default values of scrollbar attributes, as in:

```
scrollbar_get((Scrollbar)SCROLLBAR_DEFAULT,
   SCROLL_THICKNESS);
```

If your SunView application uses `scrollbar_create()`, then you may not specify the scrollbar as splittable. If you do, you will get a warning, and the `SCROLLBAR_SPLITTABLE, TRUE` attribute-value pair will be ignored. For full scrollbar functionality, use

```
xv_create(parent, SCROLLBAR, <avlist>, 0);
```

In this case, you must specify the parent (such as the panel or the canvas handle.)

Interposing on scrollbar objects to receive the `SCROLLBAR_REQUEST` event fails. The `SCROLLBAR_REQUEST` is not posted to the opaque object. You should interpose on the subwindow that the `SCROLLBAR_REQUEST` event is being posted to. Example:

```
canvas1 = xv_create(frame,CANVAS,
   WIN_PERCENT_HEIGHT,     50,
   CANVAS_AUTO_SHRINK,     FALSE,
   CANVAS_WIDTH,           700,
   CANVAS_HEIGHT,          700,
   CANVAS_REPAINT_PROC,    repaint_proc,
   0);
vert_sb1 = xv_create(canvas1,SCROLLBAR,0);
notify_interpose_event_func(
   xv_get(vert_sb1, SCROLLBAR_NOTIFY_CLIENT),
   canvas_event_proc,
   NOTIFY_SAFE);
```

Run-Time Incompatibilities

Scrollbars are now windows rather than `pixwin` regions. In particular, they now receive `WIN_REPAINT` and `WIN_RESIZE` events independent of the windows to which they are attached, and they process these events correctly. This means that clients should not call `scrollbar_paint()` when they change the size of a scrollbar or repaint a window containing scrollbars. Calling `scrollbar_paint()` will result in unnecessary flicker.

The SunView 1 routine `scrollbar_create()` is still usable, but it will not let you specify the scrollbar as splittable. For full scrollbar functionality, use `xv_create` instead.

3.13 Textsw

You cannot use `textsw_get(TEXTSW,`*attribute*`)`to get the default values for `textsw` attributes. Instead, call `textsw_get_from_defaults(`*attribute*`)`to get the attribute's default value. You can use `defaults_exists(/XView/Cmdline/`*default_name*`)` to see whether the user has overridden a default from the command line. Also, `textsw_set()` is no longer a public function. Only the textsw library can use this function. Change all `textsw_set()`, `panel_set()`, `menu_set()`, `tty_set()` calls into `xv_set()` function calls.

Ttysw: Removed Functions

Pre-SunView supported the creation of a `tty` subwindow that did not have to be a child of a frame. In XView, a `tty` subwindow is always a child of a frame. The following routines have been removed, but are documented in the *SunView 1 System Programmer's Guide* and *SunWindows Reference Manual*.

Table 3-20 *Removed Ttysw Functions*

Ttysw Function	*Comments*
`ttysw_create()`	Use `xv_create(frame, TTY)`
`ttysw_fork()`	Use `TTY_ARGV`
`ttysw_createtoolsubwindow()`	All replaced by
`ttytlsw_createtoolsubwindow()`	`xv_create()`
`ttysw_start()`	processing
`ttysw_init()`	
`ttysw_selected()`	
`ttysw_sigwinch()`	
`ttysw_handlesigwinch()`	

In addition, the unsupported but released header files `ttysw_impl.h` and `ttytlsw_impl.h` are no longer available. In particular, the `ttysw` structure has been removed. There are attributes of the `tty` object which replace most of this `structs` fields.

3.14 Windows

Apply the following rules when converting code involving windows:

1) Do not call `window_create()` with either the `WIN_HEIGHT` or `WIN_WIDTH` attributes set to less than or equal to 0. This is illegal in XView.
2) Replace calls to `win_get_pixwin(window)` with `xv_get(window, XV_SELF)`, or use the window handle itself, since this is directly passed to drawing operations.
3) Replace calls to `win_get_fd(window)` with `xv_get(window, XV_SELF)`. You can also use the window handle, since this is passed directly to most functions that required the window `fd` in SunView. (There is no access to windows by `WIN_FD` since, in a server-based system, windows are not devices. In place of `FD`-numbers, XView windows have opaque window IDs.
4) `window_fit()` causes `win` to fit its contents in the dimensions specified with `WIN_FIT_HEIGHT` and `WIN_FIT_WIDTH`.

Window Compile-Time Incompatibilities

The handling of the window tree is somewhat different in XView. Windows have links to their parent and siblings as soon as they are created, and these links persist whether a window is mapped (visible) or not. `win_insert()` and `win_remove()` are equivalent to `xv_set(window, WIN_MAP, TRUE, 0)` and `xv_set(window, WIN_MAP, FALSE, 0)`, respectively.

The following SunView routines have been removed.

Table 3-21 *Removed Low-Level Window Functions*

Low-Level Window Function	***Comments***
`win_set/getsavedrect()`	Use `xv_set/get (icon/window, WIN_RECT,...)`

`window_default_event_proc()`
`window_loop()`
`window_return()`
`window_release_event_lock()`
`win_set/getuserflags()`
	Not supported.

`win_setuserflag()`	Use `XV_KEY_DATA`
`we_set/getgfxwindow()`	`gfx_*` is totally defunct.

`win_insert/removeblanket()`
`win_isblanket()`
	No blanket windows.

`win_set/getowner()`	`xv_get(window, WIN_OWNER);`

`we_set/getparentwindow()`
`win_errorhandler()`
	Environment variables unused.

`win_set_button_order()`
`win_get_button_order()`
`win_get_scaling()`
`win_set_scaling()`
	Not supported.

Window Enumeration: Removed Functions and Procedures

The following Sunview routines that enumerate the window tree have been removed. For information on these routines refer to the *SunView 1 System Programmer's Guide.* The window enumeration routines are usually of interest to window managers, not window application programs. You can still determine a window's relatives one-by-one by using `win_getlink()`.

Table 3-22 *Removed Window Tree Enumeration Functions*

Window Tree Enumeration Function	***Comments***

`win_enumall()`
`win_enumscreen()`
`win_enumerate_children()`
`win_enumerate_subtree()`
`win_get_tree_layer()`
	Use the Xlib server call `XQueryTree()` instead.

Agents & Tiles: Removed Functions and Procedures

The following agent and tile routines have been removed.

Table 3-23 *Removed Tile/Agent Functions*

Tile/Agent Function	***Comments***
`win_register/unregister()`	
`win_set/get_flags()`	
`win_get_fd()`	Use `xv_get(window, WIN_FD)`; or just pass the window object wherever you use `WIN_FD`
`win_get_pixwin()`	Use `xv_get(window, WIN_PIXWIN)`; or just pass the window object wherever you use `WIN_PIXWIN`
`win_set/get_kbd/pick_mask()`	Use `WIN_CONSUME_EVENT`

Workstations: Removed Functions and Procedures

The following workstation routines have been removed.

Table 3-24 *Removed Workstation Functions*

Workstation Function	***Comments***

`win_setkbd()`
`win_setms()`
`win_set/remove_input_device()`
`win_is_input_device()`
`win_enum_input_device()`
`win_set/get_focus_event()`
`win_set/get_swallow_event()`
`win_release_event_lock()`
`win_set/get_event_timeout()`
	All handled (or not) by the X11/NeWS server. Refer to XView Programmers Manual

Screens: Removed Functions and Procedures

The following screen routines have been removed.

Table 3-25 *Removed Screen Functions*

Screen Function	***Comments***

`win_screennew()`
`win_initscreenfromargv()`
`win_screenget()`
`win_screendestroy()`
`win_set/getscreenpositions()`
	These are replaced by the `SERVER` and `SCREEN` objects.

Window Manager Removed Functions

The following window manager routines have been removed.

Table 3-26 *Removed Window Manager Functions*

wmgr_ Function	***Comments***
`wmgr_confirm()`	Use notices
`wmgr_winandchildrenexposed()`	Use `WIN_SHOW`
`wmgr_iswindowopen()`	Use `FRAME_CLOSED`
`wmgr_setrectalloc()`	
`wmgr_getrectalloc()`	
`win_computeclipping()`	
`win_partialrepair()`	

Window Run-Time Incompatibilities

The changes below, if not implemented could cause incompatibility problems at run-time.

Input Masks

XView does not support separate keyboard and pick input masks. For compile-time compatibility, we have defined all the window attributes dealing with keyboard and pick input masks to map to a new set of attributes which manipulate a single input mask. The following attributes therefore, have changed run-time semantics.

A sample code fragment affected by this change is listed below.

```
window_set(window,
   WIN_CONSUME_PICK_EVENTS, LOC_DRAG,0,
   WIN_CONSUME_KBD_EVENTS, WIN_NO_EVENTS,
   WIN_ASCII_EVENTS, 0, 0);
```

Table 3-27 *Changed Inputmask Attributes*

Window Attribute	*Comments*
`WIN_CONSUME_KBD_EVENTS`	Use
`WIN_CONSUME_PICK_EVENT`	`WIN_CONSUME_EVENT`
`WIN_IGNORE_KBD_EVENT`	Use
`WIN_IGNORE_PICK_EVENT`	`WIN_IGNORE_EVENT`

In SunView, this code would have resulted in `LOC_DRAG` events being enabled for the window. In XView, since the two input mask attributes map to a single input mask, specifying `WIN_NO_EVENTS` clears the single input mask and disables `LOC_DRAG` events.

Changed Window Functions

`win_setrect()` used to guarantee that the rect would change as requested, since the kernel never rejected or modified a request. The X11 window manager might alter the new size request. This means the semantics of `win_setrect()` will change. The behavior of `win_setrect()` followed by `win_getrect()` depends on external window managers. Sun will provide an external window manager which can be used with XView or other X11/NeWS applications. The XView toolkit is designed to run with every true X11window manager.

`win_setinput()`is an old, pre-SunView, SunWindows routine. The flush mask argument is ignored; no events are flushed.

`input_readevent()`no longer sets `errno` in case of error, and only returns 0 if the client code specifies non-blocking mode.

3.15 Pre-SunView Code

SunView on SunWindows does not *officially* support much of the pre-SunView functionality documented in the now-obsolete *SunWindows 2.0 Reference Manual*. However, much of this code does work in SunView, and your code may contain unsupported structures. Take the following steps to prevent unsupported structures from causing your code to fail.

Struct tool

1) Remove references to `struct tool`.
2) Replace `Tool *tool` with `Frame frame`.
3) Replace calls to `tool_parse_all()` with `xv_init(XV_ARGC_PTR_ARGV, &argc, argv, 0)`.
4) Replace call to `tool_parse_one()`with call to `xv_parse_one()`.
5) Replace call to `tool_usage()` with call to `xv_usage()`.

6) Replace calls to `tool_find_attribute(tool_attrs,` *attribute*`)` with `attr_find(tool_attrs,` *attribute*`)`.
7) Replace calls to `tool_make()`, `tool_begin()` , and `tool_create()` with `xv_create(FRAME, . . .)` For example, replace the code example below:

```
tool = tool_make(
   WIN_LABEL,               "mytool",
   WIN_ICON,                &mytool_icon,
   WIN_COLUMNS,             100,
   WIN_BOUNDARY_MGR,        True,
   WIN_ATTR_LIST,           tool_attrs,
   0);
if (tool==NULL) fprintf(stderr, "Can’t make
   tool");
```

with the following:

```
frame = xv_create(NULL, FRAME,
                  FRAME_LABEL,       "mytool",
                  WIN_COLUMNS,       100,
                  FRAME_ICON,        mytool_icon,
                  ATTR_LIST,         tool_attrs,
                  0);
if (frame==NULL) fprintf(stderr, "Can’t create
frame");
```

Note: If you are doing a complete conversion, just use the window itself in any call—there is no window `fd` in XView, so XV_SELF just returns the window.

8) Remove calls to `tool_install(tool)`and `notify_start()`. Call `xv_main_loop(frame)`instead.
9) Replace `tool_set_attribute()`with `xv_set()`
10) Instead of accessing `tool->tl_windowfd`, call `xv_get(frame, XV_SELF)` to get a window’s `fd`.
11) Remove calls to `tool_free_attribute(attrs,...)`
12) Capture `tool->tl_flags` and `TOOL_ICONIC` semantics by calling `(int)xv_get(frame, FRAME_CLOSED)` .
13) To set the icon, instead of assigning `tool->tl_icon->ic_mpr` to `ic_mpr`, call:

```
xv_set(myicon,  ICON_IMAGE, icon_image_ptr,0);
```

14) Replace calls to `win_setcursor()`by setting the window attribute `WIN_CURSOR`.

Message Subwindow

1) Since there is no `msgsw`, replace it with panels. For example, replace

```
msgsw = msgsw_create(tool, "msgsw",
   TOOL_SWEXTENDTOEDGE,TOOL_SWEXTENDTOEDGE, "",
   (struct pixfont *)0);
   if (msgsw == MSGSW_NULL)
       exit(1)
```

with

```
panel = xv_create(frame, PANEL,0);
if (panel == NULL)
   exit(1);
msg_item = xv_create(panel, PANEL_MESSAGE,
           PANEL_LABEL_STRING, "my message",
           PANEL_PAINT,        PANEL_NO_CLEAR,
           0);
if (msg_item == NULL)
   exit(1);
window_fit(panel);  /* Called to choose width
    and height */
```

2) `msgsw_setstring()` is replaced by `xv_set()` on panel message items.

3) The message subwindow stripped newline characters out of messages; you must do this yourself in panel message items.

Compile-Time Incompatibilities

The following pre-Sun functions will cause compile-time problems if not changed as described.

`gfx` Subwindow

The `gfx` subwindow no longer exists; `gfx_*` is entirely gone.

Message Subwindow

The Message Subwindow package is pre-SunView. Its functionality was replaced by the canvas, panel, or text packages in SunView (as appropriate to the usage of the Message Subwindow). The following `structs` and functions are no longer supported. (They are only documented in the *SunWindows Reference Manual*.)

Table 3-28 *Obsolete Message Subwindow Items*

Struct/Functions	***Comments***
`struct msgsubwindow` `msgsw_createtoolsubwindow()` `msgsw_create()` `msgsw_setstring()` `msgsw_display()` `msgsw_init()` `msgsw_handlesigwinch()` `msgsw_done()`	For all of these, use panel and `PANEL_MESSAGE` items instead

Empty Subwindow

The Empty Subwindow package is pre-SunView. Its functionality was replaced by the XView Canvas package. The following `structs` and functions are no longer supported. (They are only documented in the *SunWindows Reference Manual* last shipped in SunOS Release 2.0.)

Table 3-29 *Obsolete Empty Subwindow Items*

Struct/Functions	***Comments***
`struct msgsubwindow` `esw_createtoolsubwindow()` `esw_create()` `esw_setstring()` `esw_display()` `esw_init()` `esw_handlesigwinch()` `esw_done()`	For all of these, use panel and `PANEL_MESSAGE` items instead

3.16 Converting Defaults

You may wish to retain the special user-set SunView defaults in XView with the SunView `defaultsedit` program. A program named `convert_to_Xdefaults` is provided to help generate a new defaults for running under OpenWindows (and any X11 Window Systems). Any SunView defaults not known to SunView's `defaultsedit` will not be converted, and you will have to hand edit the resulting file and remove the unconverted entries. For further information on defaults, see Appendix A. Further information on xview defaults can be found in the xview reference manual page.

SunView Attributes and XView Attributes

This chapter presents an object-by-object comparison of the attributes in SunView and XView. Each list consists of a combination of SunView and XView attributes, showing the fate of each SunView attribute in XView. XView has a number of new objects, and each of them and their respective attributes are included as well. By looking over the list for each object, you can get an overview of the type and scope of changes made in each of them.

4.1 Legend

Attributes are assigned to one of five categories, each indicated by a single letter in the left column in the following tables. This way, you can see at a glance what has happened to any particular attribute, as well as get an overall view of the changes in each object. Here are the descriptions of the five categories, with an example from each one.

Unchanged Attributes

```
U   MENU_STRING
```

Unchanged attributes are those that XView retains from SunView. They have the same name and functionality in both systems.

New XView Attributes

```
N   MENU_PIN
```

New attributes either provide new features and functions or provide old ones under new names. All new attributes, including renames, comply with the *OPEN LOOK Graphical User Interface Functional Specification.*

Compatibility Attributes

```
C   MENU_CLIENT_DATA
```

This category contains SunView attributes that provide features and functions not available in XView. In general, these are attributes that deliver *non*-OPEN LOOK UI features. Unlike attributes in the following category (aliased attributes), they cannot be aliased to any new XView attributes. They are retained in this first XView release for minimal-conversion compatibility only.

‘‘Aliased’’ (Redefined) Compatibility Attributes

```
A   ICON_LABEL           ==> XV_LABEL
```

This category consists of SunView attributes that are redefined ("aliased" with `#define`) to new XView attributes. The SunView attributes are set in the left column, with the corresponding XView attributes to the right, following the ==> arrow. For a minimal conversion of a SunView application, you do not have to change these attributes. For a full conversion, rename them all to the appropriate XView attribute.

Defunct Attributes

```
D   MENU_BOXED
```

Defunct attributes are SunView attributes that were not carried over into XView. These attributes should be removed from all XView applications, including minimal conversions. The precise effect of leaving any of these attributes in your code depends on the individual attribute. For a given attribute, you will receive either a compile-time or a run-time error message complaining about its presence in your code.

4.2 Attribute Listing

In this section, attributes are listed according to the legend above. In each case, the disposition of the attribute (Unchanged, Defunct, New, etc.) appears to the left of the name of the SunView attribute. If the attribute is *aliased*, the XView attribute appears to the right, following the arrow. Packages are listed alphabetically, with attributes alphabetized within each package.

Canvas_Package

```
A   CANVAS_AUTO_CLEAR ==> OPENWIN_AUTO_CLEAR
U   CANVAS_AUTO_EXPAND
U   CANVAS_AUTO_SHRINK
D   CANVAS_FAST_MONO
U   CANVAS_FIXED_IMAGE
U   CANVAS_HEIGHT
D   CANVAS_MARGIN
N   CANVAS_MIN_PAINT_HEIGHT
N   CANVAS_MIN_PAINT_WIDTH
N   CANVAS_NTH_PAINT_WINDOW
A   CANVAS_PIXWIN     ==> CANVAS_NTH_WINDOW
U   CANVAS_REPAINT_PROC
U   CANVAS_RESIZE_PROC
U   CANVAS_RETAINED
U   CANVAS_VIEWABLE_RECT
U   CANVAS_VIEW_MARGIN
U   CANVAS_WIDTH
```

Common Attributes

See *XV_ Generic and Common Attributes*

Cursor Attributes

D	CURSOR_CROSSHAIR_BORDER_GRAVITY
D	CURSOR_CROSSHAIR_COLOR
D	CURSOR_CROSSHAIR_GAP
D	CURSOR_CROSSHAIR_LENGTH
D	CURSOR_CROSSHAIR_OP
D	CURSOR_CROSSHAIR_THICKNESS
D	CURSOR_FULLSCREEN
D	CURSOR_HORIZ_HAIR_BORDER_GRAVITY
D	CURSOR_HORIZ_HAIR_COLOR
D	CURSOR_HORIZ_HAIR_GAP
D	CURSOR_HORIZ_HAIR_LENGTH
D	CURSOR_HORIZ_HAIR_OP
D	CURSOR_HORIZ_HAIR_THICKNESS
D	CURSOR_SHOW_CROSSHAIRS
A	CURSOR_SHOW_CURSOR ==> XV_SHOW
D	CURSOR_SHOW_HORIZ_HAIR
D	CURSOR_SHOW_VERT_HAIR
D	CURSOR_VERT_HAIR_BORDER_GRAVITY
D	CURSOR_VERT_HAIR_COLOR
D	CURSOR_VERT_HAIR_GAP
D	CURSOR_VERT_HAIR_LENGTH
D	CURSOR_VERT_HAIR_OP
D	CURSOR_VERT_HAIR_THICKNESS
U	CURSOR_IMAGE
U	CURSOR_OP
U	CURSOR_XHOT
U	CURSOR_YHOT

Error Package

N	ERROR_BAD_ATTR
N	ERROR_BAD_VALUE
N	ERROR_CANNOT_GET
N	ERROR_CANNOT_SET
N	ERROR_CREATE_ONLY
N	ERROR_INVALID_OBJECT
N	ERROR_LAYER
N	ERROR_PKG

```
N   ERROR_SEVERITY
N   ERROR_STRING
N   ERROR_ERROR_PROC
```

Frame Package

```
A   FRAME_ARGC_PTR_ARGV  ==> XV_INIT_ARGC_PTR_ARGV
A   FRAME_ARGS           ==> XV_INIT_ARGS
U   FRAME_BACKGROUND_COLOR
N   FRAME_BUSY
N   FRAME_CLOSED         ==> FRAME_BASE_CLOSED
N   FRAME_CLOSED_RECT    ==> FRAME_BASE_CLOSED_RECT
N   FRAME_CMD_PANEL
N   FRAME_CMD_PUSHPIN_IN
A   FRAME_CMDLINE_HELP_PROC==> XV_USAGE_PROC
U   FRAME_CURRENT_RECT
U   FRAME_DEFAULT_DONE_PROC
U   FRAME_DONE_PROC
D   FRAME_EMBOLDEN_LABEL
U   FRAME_FOREGROUND_COLOR
N   FRAME_ICON           ==> FRAME_BASE_ICON
U   FRAME_INHERIT_COLORS
A   FRAME_LABEL          ==> XV_LABEL
N   FRAME_LEFT_FOOTER
N   FRAME_NO_CONFIRM     ==> FRAME_BASE_NO_CONFIRM
D   FRAME_NTH_WINDOW
U   FRAME_NTH_SUBFRAME
U   FRAME_NTH_SUBWINDOW
A   FRAME_OPEN_RECT      ==> XV_RECT
D   FRAME_PROPERTIES_ACTIVE
U   FRAME_PROPERTIES_PROC
D   FRAME_PROPS_ACTION_PROC
D   FRAME_PROPS_ACTIVE
D   FRAME_PROPS_APPLY PROC
N   FRAME_PROPS_PANEL    ==> FRAME_CMD_PANEL
N   FRAME_PROPS_PUSHPIN_IN==> FRAME_CMD_PUSHPIN_IN
D   FRAME_PROPS_RESET_PROC
N   FRAME_RIGHT_FOOTER
N   FRAME_SHOW_HEADER
N   FRAME_SHOW_FOOTER
U   FRAME_SHOW_LABEL
```

N	FRAME_SHOW_RESIZE_CORNER
D	FRAME_SUBWINDOWS_ADJUSTABLE

Fullscreen Package

N	FULLSCREEN_COLORMAP_WINDOW
N	FULLSCREEN_CURSOR_WINDOW
N	FULLSCREEN_INPUT_WINDOW
N	FULLSCREEN_PAINT_WINDOW
N	FULLSCREEN_RECT
N	FULLSCREEN_SYNC

Generic Attributes

See *XV__ Generic and Common Attributes*

Icon Package

A	ICON_FONT	==> XV_FONT
A	ICON_HEIGHT	==> XV_HEIGHT
U	ICON_IMAGE	
U	ICON_IMAGE_RECT	
A	ICON_LABEL	==> XV_LABEL
U	ICON_LABEL_RECT	
A	ICON_WIDTH	==> XV_WI

Menu Package

U	MENU_ACTION_IMAGE
U	MENU_ACTION_ITEM
U	MENU_ACTION_PROC
U	MENU_APPEND_ITEM
D	MENU_BOXED
D	MENU_CENTER
N	MENU_CLASS
C	MENU_CLIENT_DATA
N	MENU_COL_MAJORN
N	MENU_COL_MAJOR
U	MENU_DEFAULT
U	MENU_DEFAULT_ITEM
D	MENU_DEFAULT_SELECTION
U	MENU_DESCEND_FIRST
U	MENU_FIRST_EVENT
D	MENU_FONT
N	MENU_GEN_PIN_WINDOW
U	MENU_GEN_PROC
U	MENU_GEN_PULLRIGHT_IMAGE

U	MENU_GEN_PULLRIGHT_ITEM	
U	MENU_IMAGES	
U	MENU_IMAGE_ITEM	
D	MENU_INITIAL_SELECTION	
D	MENU_INITIAL_SELECTION_EXPANDED	
D	MENU_INITIAL_SELECTION_SELECTED	
U	MENU_INSERT	
U	MENU_INSERT_ITEM	
U	MENU_ITEM	
D	MENU_JUMP_AFTER_NO_SELECTION	
D	MENU_JUMP_AFTER_SELECTION	
U	MENU_LAST_EVENT	
D	MENU_LEFT_MARGIN	
D	MENU_MARGIN	
U	MENU_NCOLS	
U	MENU_NITEMS	
U	MENU_NOTIFY_PROC	
U	MENU_NROWS	
U	MENU_NTH_ITEM	
A	MENU_PARENT	==> XV_OWNER
N	MENU_PIN	
N	MENU_PIN_PROC	
N	MENU_PIN_WINDOW	
C	MENU_PULLRIGHT_DELTA	
U	MENU_PULLRIGHT_IMAGE	
U	MENU_PULLRIGHT_ITEM	
U	MENU_REMOVE	
U	MENU_REMOVE_ITEM	
U	MENU_REPLACE	
U	MENU_REPLACE_ITEM	
D	MENU_RIGHT_MARGIN	
U	MENU_SELECTED	
U	MENU_SELECTED_ITEM	
D	MENU_SHADOW	
D	MENU_STAY_UP	
U	MENU_STRINGS	
U	MENU_STRING_ITEM	
C	MENU_TITLE_IMAGE	
U	MENU_TITLE_ITEM	
U	MENU_TYPE	
U	MENU_VALID_RESULT	

Menu Item Package

```
U   MENU_ACTION_IMAGE
U   MENU_ACTION_ITEM
U   MENU_ACTION_PROC
U   MENU_APPEND_ITEM
D   MENU_BOXED
D   MENU_CENTER
C   MENU_CLIENT_DATA
N   MENU_COLOR
U   MENU_FEEDBACK
D   MENU_FONT
U   MENU_GEN_PROC
U   MENU_GEN_PROC_IMAGE
U   MENU_GEN_PROC_ITEM
U   MENU_GEN_PULLRIGHT
U   MENU_GEN_PULLRIGHT_IMAGE
U   MENU_GEN_PULLRIGHT_ITEM
U   MENU_IMAGE
U   MENU_IMAGE_ITEM
U   MENU_INACTIVE
U   MENU_INVERT
D   MENU_LEFT_MARGIN
D   MENU_MARGIN
N   MENU_NOTIFY_STATUS
A   MENU_PARENT         ==> XV_OWNER
U   MENU_PULLRIGHT
U   MENU_PULLRIGHT_IMAGE
U   MENU_PULLRIGHT_ITEM
U   MENU_RELEASE
U   MENU_RELEASE_IMAGE
D   MENU_RIGHT_MARGIN
D   MENU_SELECTED
U   MENU_STRING
U   MENU_STRING_ITEM
N   MENU_TITLE
U   MENU_TYPE
U   MENU_VALUE
```

Notice Package

The XView Notice Package replaces the SunView Alert package. You may find it useful to `#define` the following constants:

```
#define ALERT_YES          1
#define ALERT_NO           0
#define ALERT_FAILED      -1
#define ALERT_TRIGGERED   -2
#define alert_prompt()    notice_prompt()

A   ALERT_BUTTON         ==> NOTICE_BUTTON
D   ALERT_BUTTON_FONT
A   ALERT_BUTTON_NO      ==> NOTICE_BUTTON_NO
A   ALERT_BUTTON_YES     ==> NOTICE_BUTTON_YES
A   ALERT_MESSAGE_FONT   ==> NOTICE_FONT
A   ALERT_MESSAGE_STRINGS==> NOTICE_MESSAGE_STRINGS
A   ALERT_MESSAGE_STRINGS_ARRAY_PTR
                ==> NOTICE_MESSAGE_STRINGS_ARRAY_PTR
A   ALERT_NO_BEEPING     ==> NOTICE_NO_BEEPING
D   ALERT_OPTIONAL
D   ALERT_POSITION        (Use NOTICE_FOCUS_XY instead,
                          even through it is not API- equivalent)
D   ALERT_SREEN_CENTEREDAALERT_TRIGGER==>
              NOTICE_TRIGGER
N   NOTICE_FOCUS_XY
```

Openwin Package

`Openwin` is a new object package in XView.

```
N   OPENWIN_ADJUST_FOR_HORIZONTAL_SCROLLBAR
N   OPENWIN_ADJUST_FOR_VERTICAL_SCROLLBAR
N   OPENWIN_AUTO_CLEAR
N   OPENWIN_HORIZONTAL_SCROLLBAR
N   OPENWIN_NO_MARGIN
N   OPENWIN_NTH_VIEW
N   OPENWIN_NUMBER_OF_VIEWS
N   OPENWIN_SELECTED_VIEW
N   OPENWIN_SHOW_BORDERS
N   OPENWIN_SPLIT
N   OPENWIN_SPLIT_DIRECTION
N   OPENWIN_SPLIT_POSITION
N   OPENWIN_SPLIT_VIEW
N   OPENWIN_SPLIT_VIEW_OFFSET
```

N	OPENWIN_VERTICAL_SCROLLBAR
N	OPENWIN_VIEW_ATTRS

Panel Package

Panels are implemented in two layers. The top layer consists of a package that you use to create a generic panel object. The second layer consists of a set of panel *item* packages. You use item packages to turn a generic panel object into a panel of a particular *type*. Here is a list of the panel item packages:

```
PANEL_BUTTON
PANEL_CHECK_BOX
PANEL_CHOICE
PANEL_CHOICE_STACK
PANEL_LIST
PANEL_MESSAGE
PANEL_SLIDER
PANEL_TEXT
PANEL_TOGGLE
```

Panel Area Attributes

The Panel Area Attributes control the background or panel *area*.

U	PANEL_ACCEPT_KEYSTROKE
U	PANEL_BACKGROUND_PROC
C	PANEL_BLINK_CARET
U	PANEL_CARET_ITEM
N	PANEL_DEFAULT_ITEM
U	PANEL_EVENT_PROC
N	PANEL_EXTRA_PAINT_HEIGHT
N	PANEL_EXTRA_PAINT_WIDTH
U	PANEL_FIRST_ITEM
U	PANEL_ITEM_X_GAP
U	PANEL_ITEM_Y_GAP
D	PANEL_LABEL_BOLD
U	PANEL_LAYOUT
N	PANEL_REPAINT_PROC
D	PANEL_SHOW_MENU

Panel Generic Item Attributes*

Panel Generic Item Attributes are attributes that are *generic* to all panel-item subtypes.

* Formerly known as Generic Panel Item Attributes

Note: If your SunView code used PANEL_LABEL_IMAGE in conjunction with `panel_button_image()`, use PANEL_LABEL_STRING instead.

```
U   PANEL_ACCEPT_KEYSTROKE
A   PANEL_CLIENT_DATA    ==>XV_KEY_DATA
U   PANEL_EVENT_PROC
N   PANEL_ITEM_CLASS
N   PANEL_ITEM_COLOR
U   PANEL_ITEM_RECT
A   PANEL_ITEM_X         ==> XV_X
A   PANEL_ITEM_Y         ==> XV_Y
C   PANEL_LABEL_BOLD
C   PANEL_LABEL_FONT
C   PANEL_LABEL_IMAGE
U   PANEL_LABEL_STRING
N   PANEL_LABEL_WIDTH
U   PANEL_LABEL_X
U   PANEL_LABEL_Y
U   PANEL_LAYOUT
D   PANEL_MENU_CHOICE_FONTS
D   PANEL_MENU_CHOICE_IMAGES
D   PANEL_MENU_CHOICE_STRINGS
D   PANEL_MENU_CHOICE_VALUES
D   PANEL_MENU_TITLE_FONT
D   PANEL_MENU_TITLE_IMAGE
D   PANEL_MENU_TITLE_STRING
U   PANEL_NEXT_ITEM
U   PANEL_NOTIFY_PROC
N   PANEL_NOTIFY_STATUS
U   PANEL_PAINT
A   PANEL_PARENT_PANEL   ==> XV_OWNER
N   PANEL_REPAINT_PROC
A   PANEL_SHOW_ITEM      ==> XV_SHOW
D   PANEL_SHOW_MENU
U   PANEL_VALUE_X
U   PANEL_VALUE_Y
```

Panel Choice and Toggle Attributes

```
C   PANEL_CHOICES_BOLD
N   PANEL_CHOICE_COLOR
C   PANEL_CHOICE_FONT
C   PANEL_CHOICE_FONTS
U   PANEL_CHOICE_IMAGE
U   PANEL_CHOICE_IMAGES
```

N	PANEL_CHOICE_NCOLS
N	PANEL_CHOICE_NROWS
U	PANEL_CHOICE_STRING
U	PANEL_CHOICE_STRINGS
N	PANEL_CHOICE_UP_ARROW
C	PANEL_CHOICE_X
C	PANEL_CHOICE_XS
C	PANEL_CHOICE_Y
C	PANEL_CHOICE_YS
N	PANEL_CHOOSE_ONE
N	PANEL_CHOOSE_NONE
C	PANEL_CYCLE
N	PANEL_DEFAULT_VALUE
U	PANEL_DISPLAY_LEVEL
U	PANEL_FEEDBACK
U	PANEL_LAYOUT
C	PANEL_MARK_IMAGE
C	PANEL_MARK_IMAGES
C	PANEL_MARK_X
C	PANEL_MARK_XS
C	PANEL_MARK_Y
C	PANEL_MARK_YS
D	PANEL_MENU_MARK_IMAGE
D	PANEL_MENU_NOMARK_IMAGE
C	PANEL_NOMARK_IMAGE
C	PANEL_NOMARK_IMAGES
D	PANEL_SHOW_MENU_MARK
U	PANEL_TOGGLE_VALUE
U	PANEL_VALUE

Note: SunView's `PANEL_CYCLE` item is retained for compatibility only. What you really get is a `PANEL_CHOICE_STACK`. (The OPEN LOOK UI description is *Abbreviated Menu Button*.) In the SunView functionality, which no longer exists, pressing the left mouse button on the appropriate glyph toggled the options; in XView, the options are selected from a menu.

Panel Button Attributes

In XView, panel buttons are implemented with their own panel-item subtype.

```
PANEL_INACTIVE
PANEL_ITEM_MENU
```

Panel List Attributes

Panel lists are implemented with a new panel-item subtype in XView.

N	PANEL_LIST_CLIENT_DATA
N	PANEL_LIST_CLIENT_DATAS

N	PANEL_LIST_DELETE
N	PANEL_LIST_DISPLAY_ROWS
N	PANEL_LIST_FONT
N	PANEL_LIST_FONTS
N	PANEL_LIST_GLYPH
N	PANEL_LIST_GLYPHS
D	PANEL_LIST_HEIGHT
N	PANEL_LIST_INSERT
N	PANEL_LIST_NROWS
N	PANEL_LIST_ROW_HEIGHT
N	PANEL_LIST_SELECT
N	PANEL_LIST_STRING
N	PANEL_LIST_STRINGS
N	PANEL_LIST_WIDTH

Panel Message Attributes

In XView, panel messages are implemented with a panel-item subtype, which currently consists of a single compatibility attribute.

U	PANEL_LABEL_BOLD

Panel Slider Attributes

U	PANEL_MAX_VALUE
U	PANEL_MIN_VALUE
U	PANEL_NOTIFY_LEVEL
U	PANEL_SHOW_RANGE
U	PANEL_SHOW_VALUE
N	PANEL_DIRECTION
N	PANEL_SLIDER_END_BOXES
N	PANEL_TICKS
U	PANEL_SLIDER_WIDTH
U	PANEL_VALUE
D	PANEL_VALUE_FONT

Panel Text Attributes

U	PANEL_MASK_CHAR
N	PANEL_LABEL_WIDTH
U	PANEL_NOTIFY_LEVEL
U	PANEL_VALUE
U	PANEL_VALUE_DISPLAY_LENGTH
D	PANEL_VALUE_FONT
U	PANEL_VALUE_STORED_LENGTH
N	PANEL_VALUE_UNDERLINED

Screen Package

In XView, a display screen is treated as an object in its own right.

```
N   SCREEN_IMAGE_DEPTH
N   SCREEN_IMAGE_BITS
N   SCREEN_NUMBER
N   SCREEN_SERVER
```

Scrollbar Package

Note:
`SCROLLBAR_SPLITTABLE` is ignored if called by `scrollbar_create()`. Use `xv_create` instead.

```
N   SCROLLBAR_COMPUTE_SCROLLBAR_PROC
N   SCROLLBAR_MENU
N   SCROLLBAR_NORMALIZE_PROC
N   SCROLLBAR_OVERSCROLL
N   SCROLLBAR_PAGE_LENGTH
N   SCROLLBAR_PIXELS_PER_UNIT
N   SCROLLBAR_SPLITTABLE
D   SCROLL_ABSOLUTE_CURSOR
D   SCROLL_ACTIVE_CURSOR
D   SCROLL_ADVANCED_MODE
D   SCROLL_BACKWARD_CURSOR
D   SCROLL_BAR_COLOR
D   SCROLL_BAR_DISPLAY_LEVEL
D   SCROLL_BORDER
D   SCROLL_BUBBLE_COLOR
D   SCROLL_BUBBLE_DISPLAY_LEVEL
D   SCROLL_BUBBLE_MARGIN
D   SCROLL_DIRECTION
             ==>N SCROLLBAR_DIRECTION
D   SCROLL_END_POINT_AREA
D   SCROLL_FORWARD_CURSOR
D   SCROLL_GAP
D   SCROLL_HEIGHT
D   SCROLL_LAST_VIEW_START
             ==>N SCROLLBAR_LAST_VIEW_START
D   SCROLL_LEFT
D   SCROLL_LINE_HEIGHT
             ==>N SCROLLBAR_PIXELS_PER_UNIT
D   SCROLL_MARGIN
D   SCROLL_MARK
D   SCROLL_NORMALIZE
D   SCROLL_NOTIFY_CLIENT
             ==>N SCROLLBAR_NOTIFY_CLIENT
D   SCROLL_OBJECT
```

```
D   SCROLL_OBJECT_LENGTH
                     ==>N SCROLLBAR_OBJECT_LENGTH
D   SCROLL_PAGE_BUTTONS
D   SCROLL_PAGE_BUTTON_LENGTH
D   SCROLL_PAINT_BUTTONS_PROC
D   SCROLL_PIXWIN
D   SCROLL_PLACEMENT
D   SCROLL_RECT
D   SCROLL_REPEAT_TIME
D   SCROLL_REQUEST_MOTION
D   SCROLL_REQUEST_OFFSET
D   SCROLL_THICKNESS
D   SCROLL_TOP
D   SCROLL_TO_GRID
D   SCROLL_VIEW_LENGTH   ==>N SCROLLBAR_VIEW_LENGTH
D   SCROLL_VIEW_START    ==>N SCROLLBAR_VIEW_START
D   SCROLL_WIDTH
```

Selection Package

A number of SunView selection procedures have been converted to attributes.

```
N   SELN_REQ_BYTESIZE
N   SELN_REQ_COMMIT_PENDING_DELETE
N   SELN_REQ_CONTENTS_ASCII
N   SELN_REQ_CONTENTS_PIECES
N   SELN_REQ_DELETE
N   SELN_REQ_END_REQUEST
N   SELN_REQ_FAILED
N   SELN_REQ_FAKE_LEVEL
N   SELN_REQ_FILE_NAME
N   SELN_REQ_FIRST
N   SELN_REQ_FIRST_UNIT
N   SELN_REQ_LAST
N   SELN_REQ_LAST_UNIT
N   SELN_REQ_LEVEL
N   SELN_REQ_RESTORE
N   SELN_REQ_SET_LEVEL
N   SELN_REQ_YIELD
```

Server Package

Like the screen, the server is treated as an object in XView.

```
N   SERVER_ASCII_KEYSTATE
N   SERVER_ASCII_TO_KEYCODE_MAP
N   SERVER_BANG
N   SERVER_DO_DRAG_MOVE
N   SERVER_EXTENSION_PROC
N   SERVER_FONT_WITH_NAME
N   SERVER_IMAGE_DEPTH
N   SERVER_IMAGE_BITS
N   SERVER_JOURNALLING
N   SERVER_JOURNAL_SYNC_ATOM
N   SERVER_JOURNAL_SYNC_EVENT
N   SERVER_KEY_EVENTS_MAP
N   SERVER_NONASCII_KEYSTATE
N   SERVER_NTH_SCREEN
N   SERVER_SEMANTIC_MAP
N   SERVER_SYNC
N   SERVER_SYNC_AND_PROCESS_EVENTS
N   SERVER_XV_MAP
N   SERVER_WM_APPLY_PROPERTIES
N   SERVER_WM_CONFIGURE_DENIED
N   SERVER_WM_DECORATION_HINTS
N   SERVER_WM_DISMISS
N   SERVER_WM_LEFT_FOOTER
N   SERVER_WM_PUSHPIN_STATE
N   SERVER_WM_RESCALE
N   SERVER_WM_RESCALE_STATE
N   SERVER_WM_RESET_PROPERTIES
N   SERVER_WM_RIGHT_FOOTER
N   SERVER_WM_SHOW_PROPERTIES
N   SERVER_WM_STATE
N   SERVER_WM_TAKE_FOCUS
N   SERVER_WM_WINDOW_BUSY
N   SERVER_WM_WINDOW_MOVED
N   XV_HEIGHT
N   XV_WIDTH
```

Text Subwindow Package

D	TEXTSW_ADJUST_IS_PENDING_DELETE
U	TEXTSW_AGAIN_RECORDING
U	TEXTSW_AUTO_INDENT
U	TEXTSW_AUTO_SCROLL_BY
U	TEXTSW_BLINK_CARET
U	TEXTSW_BROWSING
U	TEXTSW_CHECKPOINT_FREQUENCY
U	TEXTSW_CLIENT_DATA
U	TEXTSW_CONFIRM_OVERWRITE
U	TEXTSW_CONTENTS
U	TEXTSW_CONTROL_CHARS_USE_FONT
U	TEXTSW_DISABLE_CD
U	TEXTSW_DISABLE_LOAD
U	TEXTSW_EDIT_COUNT
U	TEXTSW_FILE_CONTENTS
U	TEXTSW_FIRST
U	TEXTSW_FIRST_LINE
U	TEXTSW_HISTORY_LIMIT
U	TEXTSW_INSERTION_POINT
U	TEXTSW_INSERT_FROM_FILE
U	TEXTSW_INSERT_MAKES_VISIBLE
U	TEXTSW_LENGTH
U	TEXTSW_LINE_BREAK_ACTION
U	TEXTSW_LOWER_CONTEXT
U	TEXTSW_MEMORY_MAXIMUM
U	TEXTSW_MULTI_CLICK_SPACE
U	TEXTSW_MULTI_CLICK_TIMEOUT
U	TEXTSW_NOTIFY_PROC
U	TEXTSW_READ_ONLY
U	TEXTSW_STATUS
U	TEXTSW_STORE_CHANGES_FILE
N	TEXTSW_SUBMENU_EDIT
N	TEXTSW_SUBMENU_FIND
N	TEXTSW_SUBMENU_VIEW
U	TEXTSW_STORE_SELF_IS_SAVE
U	TEXTSW_UPDATE_SCROLLBAR
U	TEXTSW_UPPER_CONTEXT
N	TEXTSW_WRAPAROUND_SIZE

TTY Package

U	TTY_ARGV
U	TTY_CONSOLE
U	TTY_PAGE
U	TTY_QUIT_ON_CHILD_DEATH

XV_ Generic and Common Attributes

In XView, there is a set of attributes that you use to control features shared by (a) all XView objects or (b) many XView objects. Attributes that apply to any object are called *generic* attributes; they have `Gen` beside them in the table below. Attributes that apply to a number of objects are called *common attributes*, and are preceded with `Com`. All have the prefix `XV_`.

N	Gen	HELP_STRING_FILENAME
N	Gen	XV_AUTO_CREATE
N	Com	XV_BOTTOM_MARGIN
N	Gen	XV_COPY
N	Gen	XV_COPY_OF
N	Com	XV_DEVICE_NAME
N	Com	XV_DEVICE_NUMBER
N	Gen	XV_END_CREATE
N	Com	XV_FONT
N	Com	XV_HEIGHT
N	Gen	XV_HELP
N	Gen	XV_HELP_DATA
N	Gen	XV_IS_SUBTYPE_OF
N	Gen	XV_KEY_DATA
N	Gen	XV_KEY_DATA_COPY_PROC
N	Gen	XV_KEY_DATA_REMOVE
N	Gen	XV_KEY_DATA_REMOVE_PROC
N	Gen	XV_LABEL
N	Com	XV_LEFT_MARGIN
N	Com	XV_MARGIN
N	Gen	XV_NAME
N	Gen	XV_OWNER
N	Com	XV_RECT
N	Gen	XV_REF_COUNT
N	Com	XV_RIGHT_MARGIN
N	Com	XV_ROOT
N	Gen	XV_SELF
N	Com	XV_SHOW
N	Gen	XV_STATUS
N	Gen	XV_STATUS_PTR

N	Com	XV_TOP_MARGIN
N	Gen	XV_TYPE
D	Com	XV_VISUAL
N	Com	XV_WIDTH
N	Com	XV_X
N	Gen	XV_XID
N	Com	XV_Y

Window Package

N	WIN_ALARM	
N	WIN_ALARM_DATA	
U	WIN_BELOW	
N	WIN_BIT_GRAVITY	
C	WIN_BOTTOM_MARGIN	
C	WIN_CLIENT_DATA	
U	WIN_COLUMNS	
A	WIN_COLUMN_GAP	==> WIN_COL_GAP
A	WIN_COLUMN_WIDTH	==> WIN_COL_WIDTH
N	WIN_COL_GAP	
N	WIN_COL_WIDTH	
N	WIN_CONSUME_EVENT	
N	WIN_CONSUME_EVENTS	
C	WIN_CONSUME_KBD_EVENT	
C	WIN_CONSUME_KBD_EVENTS	
C	WIN_CONSUME_PICK_EVENT	
C	WIN_CONSUME_PICK_EVENTS	
U	WIN_CURSOR	
N	WIN_DEPTH	
N	WIN_DESIRED_HEIGHT	
N	WIN_DESIRED_WIDTH	
C	WIN_DEVICE_NAME	
C	WIN_DEVICE_NUMBER	
U	WIN_ERROR_MSG	
U	WIN_EVENT_PROC	
D	WIN_EVENT_STATE	
C	WIN_FD	
U	WIN_FIT_HEIGHT	
U	WIN_FIT_WIDTH	
C	WIN_FONT	
N	WIN_FRAME	
U	WIN_GRAB_ALL_INPUT	

```
C   WIN_HEIGHT
U   WIN_HORIZONTAL_SCROLLBAR
N   WIN_IGNORE_EVENT
N   WIN_IGNORE_EVENTS
C   WIN_IGNORE_KBD_EVENT
C   WIN_IGNORE_KBD_EVENTS
C   WIN_IGNORE_PICK_EVENT
C   WIN_IGNORE_PICK_EVENTS
D   WIN_INPUT_DESIGNEE
N   WIN_INPUT_MASK
N   WIN_INPUT_ONLY
D   WIN_IN_TRANSIT_EVENTS
U   WIN_KBD_FOCUS
A   WIN_KBD_INPUT_MASK       ==> WIN_INPUT_MASK
C   WIN_LEFT_MARGIN
N   WIN_MAP
U   WIN_MENU
U   WIN_MOUSE_XY
C   WIN_NAME
N   WIN_NO_DECORATIONS
A   WIN_NOTIFY_EVENT_PROC ==>
             WIN_NOTIFY_SAFE_EVENT_PROC
N   WIN_NOTIFY_IMMEDIATE_EVENT_PROC
C   WIN_OWNER
N   WIN_PARENT
U   WIN_PERCENT_HEIGHT
U   WIN_PERCENT_WIDTH
A   WIN_PICK_INPUT_MASK   ==> WIN_INPUT_MASK
C   WIN_PIXWIN
A   WIN_RECT          ==> XV_RECT
N   WIN_RETAINED
C   WIN_RIGHT_MARGIN
U   WIN_RIGHT_OF
U   WIN_ROWS
U   WIN_ROW_GAP
U   WIN_ROW_HEIGHT
U   WIN_SCREEN_RECT
U   WIN_SHOW
N   WIN_TOP_LEVEL
C   WIN_TOP_MARGIN
N   WIN_TRANSPARENT
```

D	WIN_TYPE
U	WIN_VERTICAL_SCROLLBAR
C	WIN_WIDTH
N	WIN_WINDOW_GRAVITY
C	WIN_X
C	WIN_Y

New Features

This section describes new XView features. Some of these features provide similar functionality to SunView, through a different mechanism; others are only possible in a networked, server-based environment.

5.1 The Generic Object

Every XView object is created using the new Generic Object package. This allows clients to use a single `create`, `set`, `get`, and `destroy` interface. The new functions are listed below.

Table 5-1 *Generic Functions*

Generic Function	***Comments***
`xv_create()`	create any object.
`xv_set()`	set attributes.
`xv_find()`	locate an object; if unsuccessful, revert to `xv_create()`.
`xv_get()`	get value of attribute.
`xv_destroy()`	destroy any object.
`xv_init()`	initialize the notifier, read passed attributes, initialize defaults/resource-manager database, load server resource-manager database, and read ~/.Xdefaults database.

For compatibility, the SunView individual object routines, such as `window_create()` and `icon_create()`, are provided. These routines in turn call `xv_create/xv_set/xv_get/xv_destroy()` as appropriate. Since all XView objects are at least generic objects, generic attributes can be applied to any object.

Arbitrary Key Data Storage

Two generic attributes provide the client with arbitrary key-value storage on any object. The generic object package manages an association table for each object, which you can use to define attributes peculiar to your application.

5.2 Other Objects

Server Objects

Server objects allow one XView client to create windows and operate on multiple X11/NeWS servers simultaneously. This was not possible in SunView. There are attributes to enumerate the list of screen objects attached to each server.

Screen Objects

Screen objects let XView clients create windows on several screens on one server; this is possible in SunView using low-level calls, but very difficult.

5.3 Sample Client Code

The simplest XView application is similar to the SunView style.

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

main(argc, argv)
    int   argc;
    char  **argv;
{
    Frame frame1;
    Panel panel1;

   /* Create a frame with a panel specifying a root
    * window of NULL defaults to the root window on
    * the default screen of the default server. */
   frame1 = xv_create(NULL, FRAME, 0);
   panel1 = xv_create(frame1, PANEL, 0);
```

Creating Objects in a Multiple Server/Screen Environment

Cursors and menus can be created without specifying an owner; it is only when the cursor or menu is actually rendered that it is bound to a particular server or screen.

On the other hand, frames, windows, and fonts are bound to a particular display when they are created. If you do not specify a particular server or screen, they are bound to the default server/screen. If you later try to use them on another server or screen, the attempt will fail.

Here is an example of an XView program that creates and controls windows on different screens attached to a single server. This example demonstrates how to create windows on screen # 0 and screen # 1:

```
#include <xview/xview.h>
#include <xview/text.h>

Frame      frame_0, frame_1;
Textsw     textsw_0, textsw_1;
Xv_Window  win_0, win_1;
Xv_Screen  screen_0, screen_1;
Xv_Server  screen_0, screen_1;

main(argc,argv)
int    argc;
char   *argv[];
{
    my_server = xv_init(XV_INIT_ARGC_PTR_ARGV,&argc,argv,0);
    screen_0=(my_server) xv_get(my_server,
                SERVER_NTH_SCREEN,    0);
    win_0 = (Xv_Window) xv_get( screen_0, XV_ROOT );
    screen_1=(Xv_Screen) xv_get( my_server,
                SERVER_NTH_SCREEN,    1);
    win_1 = (Xv_Window) xv_get( screen_1, XV_ROOT );

    frame_0 = (Frame) xv_create( win_0, FRAME,
                FRAME_LABEL,           "SCREEN 0",
                0 );
    textsw_0 = (Textsw) xv_create( frame_0, TEXTSW,
                TEXTSW_BLINK_CARET,    FALSE,
                TEXTSW_CONTENTS,       "Textsw on Screen 0",
                TEXTSW_MEMORY_MAXIMUM, TEXTSW_INFINITY,
                0 );
    frame_1 = (Frame) xv_create( win_1, FRAME,
                FRAME_LABEL,           "SCREEN 1",
                XV_SHOW,               TRUE
                0 );
    textsw_1 = (Textsw) xv_create( frame_1, TEXTSW,
                TEXTSW_BLINK_CARET,    FALSE,
                TEXTSW_CONTENTS,       "Textsw on Screen 1",
                TEXTSW_MEMORY_MAXIMUM, TEXTSW_INFINITY,
                0 );
    xv_main_loop( frame_0 );
}
```

5.4 Initialization

XView command-line options are loaded into the in-memory defaults database when they are parsed. To see whether the user tried to override some particular setting, you no longer need to pre-scan the command line yourself; instead call `default_exists()` on command line options to see whether the option was set from the command line. For example:

```
if (defaults_exists("xview.Cmdline.Columns",0)) {
    /* user specified -Ww */
}
```

5.5 Input

New input-related window attributes `WIN_META_EVENTS` and `WIN_UP_META_EVENTS` let you enable `<Meta>` key press and release events. You can also inquire directly as to the state of the mouse buttons from any input event you receive, with the new event state macros, `event_left_is_down()`, `event_middle_is_down()`, `event_right_is_down()`, and `event_button_is_down()`. You can now get as well as set `WIN_MOUSE_XY.`

5.6 Server Image

The X11/NeWS server provides the capability of creating bitmap objects that are stored in the server. This reduces the memory usage in the client's process and reduces the communication overhead to the server. To take advantage of this, XView provides a `SERVER_IMAGE` object (essentially a memory pixrect whose bitmap is stored in the server).

A

Defaults

This appendix contains a comprehensive listing of XView defaults. Further information is available from the XView reference manual page. Please note that arguments to `defaults_get_*` have changed.

A.1 SunView and XView Defaults

This section lists SunView and XView defaults in tabular form.

Table A-1 *SunView vs. XView Defaults*

SunView Default	***XView Default***
/defaults/read_defaults_database	(n/a in the OPEN LOOK UI: use Factory/Custom switch in Props)
/sunview/scale	Window.Scale
/sunview/font	Font.Name
/sunview/cmdline/label	Window.Header
n/a	Window.Footer
/sunview/cmdline/columns	Window.Columns
/sunview/cmdline/rows	Window.Rows
/sunview/cmdline/width	Window.Width
/sunview/cmdline/height	Window.Height
/sunview/cmdline/x	Window.X
/sunview/cmdline/y	Window.Y
/sunview/cmdline/iconic	Window.Iconic
/sunview/cmdline/no_name_stripe	(n/a in the OPEN LOOK UI: always FALSE)
/sunview/cmdline/set_default_color	Window.InheritColor
/sunview/cmdline/foreground_color	Window.Color.Foreground

SunView Default	***XView Default***
/sunview/cmdline/background_color	Window.Color.Background
/sunview/cmdline/icon.font	Icon.Font.Name
/sunview/cmdline/icon.image	Icon.Pixmap
n/a	Icon.Header
/sunview/cmdline/icon.label	Icon.Footer
/sunview/cmdline/icon.x	Icon.X
/sunview/cmdline/icon.y	Icon.Y
/text/edit_back_char	DEL or Backspace
/text/edit_back_word	Ctrl-w
/text/edit_back_line	Ctrl-u
/text/multi_click_timeout	Mouse.Multiclick.Timeout
/sunview/cmdline/server	Server.Name
/tty/checkpoint_frequency	Term.CheckpointFrequency
/tty/text_wraparound_size	Term.MaxLogFileSize
/text/scratch_window	(n/a in the OPEN LOOK UI: always FALSE)
/text/scrollable	Text.EnableScrollbar
/text/adjust_is_pending_delete	(n/a in the OPEN LOOK UI: always TRUE)
/text/again_limit	Text.AgainLimit
/text/auto_indent	Text.AutoIndent
/text/auto_scroll_by	Text.AutoScrollBy
/text/blink_caret	Text.BlinkCaret
/text/checkpoint_frequency	Text.CheckpointFrequency
/text/confirm_overwrite	Text.ConfirmOverwrite
/text/control_chars_use_font	Text.DisplayControlChars
/text/font	Font.Name
/text/history_limit	Text.UndoLimit
/text/insert_makes_caret_visible	Text.InsertMakesCaret Visible
/text/long_line_break_mode	Text.LineBreak

SunView Default	***XView Default***
/text/load_file_of_directory	(delete: always is ‘‘set directory’’)
/text/lower_context	Text.Margin.Bottom
/text/memory_maximum	Text.MaxDocumentSize
/text/multi_click_space	Mouse.Multiclick.Space
/text/multi_click_timeout	Mouse.Multiclick.Timeout
SunView Default	XView Default
/text/store_changes_file	Text.StoreChangesFile
/text/store_self_is_save	(delete: always is TRUE)
/text/upper_context	Text.Margin.Top
/text/left_margin	Text.Margin.Left
/text/right_margin	Text.Margin.Right
/text/tab_width	Text.TabWidth
/text/extras_menu_filename	Text.ExtrasMenuFilename
/text/retained	Text.Retained
/text/contents	(n/a in the OPEN LOOK UI: no scratch window)
/input/arrow_Keys	(n/a in the OPEN LOOK UI: not a Kernel-based window system)
/input/left_Handed	(n/a in the OPEN LOOK UI: not a Kernel-based window system)
/tty/auto_indent	Text.AutoIndent
/tty/control_chars_use_font	Text.DisplayControlChars
/tty/insert_makes_caret_visible	Text.InsertMakes CaretVisible
/tty/append_only_log	Term.EnableEdit
/tty/bold_style	Term.BoldStyle
/tty/inverse_mode	Term.InverseStyle
/tty/underline_mode	Term.UnderlineStyle
/sunview/audible_bell	Alarm.Audible

SunView Default	***XView Default***
/sunview/visible_bell	Alarm.Visible
/sunview/alert_jump_cursor	Notice.JumpCursor
/sunview/alert_bell	Notice.BeepCount

Table A-2 *Changes in User-Settable Defaults*

SunView1 1.75	***XView/OpenWindows***
`/sunview/scale`	`Window.Scale:`
`/Compatibility`	`<defunct>`
`/compatibility`	`<defunct>`
`/Scrollbar`	`<defunct>`
`/Defaults`	`<defunct>`
`/Indent`	`<defunct>`
`/Input`	`<defunct>`
`/Mail`	`<defunct>`
`/Menu/Pullright_delta`	`OpenWindow.DragRightDistance:`
`/Menu`	`<all other /Menu defaults defunct>`
`/Sunview/Audible_bell`	`Alarm.Audible:`
`/Sunview/Visible_bell`	`Alarm.Visible:`
`/Sunview/Alert_jump_cursor`	`Notice.JumpCursor:`
`/Sunview/Alert_bell`	`Notice.BeepCount:`
`/Sunview`	`<all other`
`/Sunview`	`defaults defunct>`
`/Tty/Auto_indent`	`Text.AutoIndent:`
`/Tty/Control_chars_use_font`	`Text.DisplayControlChars:`
`/Tty/control_chars_use_font`	`Text.DisplayControlChars:`
`/Tty/Insert_makes_caret_visible`	`Text.InsertMakesCaretVisible:`
`/Tty/insert_makes_caret_visible`	`Text.InsertMakesCaretVisible:`
`/Tty/Append_only_log`	`Term.EnableEdit:`
`/Tty/Bold_style`	`Term.BoldStyle:`
`/Tty/Inverse_mode`	`Term.InverseStyle:`
`/Tty/Underline_mode`	`Term.UnderlineStyle:`
`/Text/Edit_back_char`	`<defunct>`
`/Text/Edit_back_word`	`<defunct>`
`/Text/Edit_back_line`	`<defunct>`
`/Text/Multi_click_timeout`	`OpenWindows.MultiClickTimeout:` `<now in tenths of seconds>`
`/Tty/Checkpoint_frequency`	`Term.CheckpointFrequency:`
`/Tty/Text_wraparound_size`	`Term.MaxLogFileSize:`
`/Text/Scratch_window`	`<defunct>`

SunView1 1.75	***XView/OpenWindows***
`/Text/Scrollable`	`Text.EnableScrollbar:`
`/Text/Adjust_is_pending_delete`	`<defunct>`
`/Text/Again_limit`	`Text.AgainLimit:`
`/Text/Auto_indent`	`Text.AutoIndent:`
`/Text/Auto_scroll_by`	`Text.AutoScrollBy:`
`/Text/Blink_caret`	`Text.BlinkCaret:`
`/Text/Checkpoint_frequency`	`Text.CheckpointFrequency:`
`/Text/Confirm_overwrite`	`Text.ConfirmOverwrite:`
`/Text/Control_chars_use_font`	`Text.DisplayControlChars:`
`/Text/control_chars_use_font`	`Text.DisplayControlChars:`
`/Text/Font`	`<defunct>`
`/Text/Insert_makes_caret_visible`	`Text.InsertMakesCaretVisible:`
`/Text/Long_line_break_mode`	`Text.LineBreak:`
`/Text/long_line_break_mode`	`Text.LineBreak:`
`/Text/Load_file_of_directory`	`<defunct>`
`/Text/Lower_context`	`Text.Margin.Bottom:`
`/Text/Memory_maximum`	`Text.MaxDocumentSize:`
`/Text/Multi_click_space`	`Mouse.Multiclick.Space:`
`/Text/Multi_click_timeout`	`Mouse.Multiclick.Timeout:`
`/Text/Store_changes_file`	`Text.StoreChangesFile:`
`/Text/Store_self_is_save`	`<defunct>`
`/Text/Upper_context`	`Text.Margin.Top:`
`/Text/Left_margin`	`Text.Margin.Left:`
`/Text/Right_margin`	`Text.Margin.Right:`
`/Text/Tab_width`	`Text.TabWidth:`
`/Text/Extras_menu_filename`	`Text.ExtrasMenuFilename:`
`/Text/Retained`	`Text.Retained:`
`/Text/Contents`	`<defunct>`
`/Input/Keymap_Directory`	`<defunct>`

Changes to the `defaults.h` interface in XView

The following entries comprise the latest changes to the `defaults.h` interface. In each case, the first entry is the SunView interface and the second is the XView interface, if it exists. New interfaces are not listed in this section except for compatibility replacement.

A note on terminology:

Resource is an X11 term that means ‘‘the string stored in the defaults database that describes the default.’’ Lines in the `.Xdefaults` database, for instance, take the form:

```
resource:    value
```

There are two kinds of resource: *instance* and *class*. An instance resource specifies a specific default for a specific application, such as `cmdtool.promptString`. A class resource specifies a generic default, covering a class of applications, such as `Editor.AutoIndent`. Instances start each key with lowercase (with the ‘‘glommed’’ words separated by dots), and classes start each key with an uppercase character.

Further information is available from the XView reference manual page.

Table A-3 *Changes to the defaults.h Interface*

SunView/XView Interface **Comments**

```
============================================================================
defaults_exists(path_name, status)
        char    *path_name;           /* Node name to test for existence */
        int     *status;                  /* Status flag */

defaults_exists(name, class)
        char    *name;
        char    *class;

============================================================================
struct _default_pairs {
        char    *name;                     /* Name of pair */
        int     value;                   /* Value of pair */
};
typedef struct _default_pairs Defaults_pairs;

typedef struct _default_pairs {
        char    *name;                     /* Name of pair */
        int     value;                   /* Value of pair */
} Defaults_pairs;

============================================================================
defaults_get_boolean(path_name, default_bool, status)
        char    *path_name;                   /* Path name */
        Bool    default_bool;            /* Default value */
        int     *status;                   /* Status flag */

defaults_get_boolean(name, class, default_bool)
        char    *name;
        char    *class;
        Bool    default_bool;            /* Default value */

============================================================================
defaults_get_character(path_name, default_character, status)
        char    *path_name;            /* Full database node path name */
        char    default_character;     /* Default return value */
        int     *status;                 /* Status flag */

defaults_get_character(name, class, default_character)
        char    *name;
        char    *class;
        char    default_character;     /* Default return value */
============================================================================
```

SunView/XView Interface **Comments**

```
============================================================================
defaults_get_child()                                       --gone
============================================================================
defaults_get_enum(path_name, pairs, status)
        char            *path_name;     /* Full database path name */
        Defaults_pairs  *pairs;         /* Pairs table */
        int             *status;          /* Status flag */

defaults_get_enum(name, class, pairs)                       --new
        char    *name;
        char    *class;
        Defaults_pairs  *pairs;           /* Pairs table */
============================================================================
defaults_get_enumeration()              NOT IMPLEMENTED!!!
in xview's defaults.h, but not implemented (probably defunct)
============================================================================
defaults_get_integer(path_name, default_integer, status)
        char            *path_name;     /* Full database node name */
        int             default_integer; /* Default return value */
        int             *status;          /* Status flag */

defaults_get_integer(name, class, default_integer)
        char    *name;
        char    *class;
        int      default_integer; /* Default return value */
============================================================================
defaults_get_integer_check(path_name, default_int, minimum, maximum, status)
        char    *path_name;           /* Full path name of node */
        int     default_int;          /* Default return value */
        int     minimum;                /* Minimum value */
        int     maximum;                /* Maximum value */
        int     *status;                  /* Status flag */

defaults_get_integer_check(name, class, default_int, minimum, maximum)
        char    *name;
        char    *class;
        int     default_int;          /* Default return value */
        int     minimum;                /* Minimum value */
        int     maximum;                /* Maximum value */
============================================================================
defaults_get_sibling()          NOT SUPPORTED IN XView!!!!
============================================================================
```

SunView/XView Interface **Comments**

```
============================================================================
defaults_get_string(path_name, default_string, status)
        char            *path_name;     /* Full database node name */
        char            *default_string; /* Default return value */
        int             *status;          /* Status flag */

defaults_get_string(name, class, default_string)
        char    *name;
        char    *class;
        char    *default_string;      /* Default return value */
============================================================================
defaults_get_default()               NOT SUPPORTED IN XView!!!
============================================================================
defaults_init(read_defaults_database)
   int          read_defaults_database;

defaults_init_db()                                --NEW INTERFACE
============================================================================
defaults_load_db(filename)           --NEW INTERFACE in XView
        char    *filename;
============================================================================
defaults_store_db(filename)          --NEW INTERFACE in XView
        char    *filename;
============================================================================
defaults_lookup(name, pairs)                          --NO CHANGE
   register char*name;               /* Name to look up */
   register Defaults_pairs*pairs;     /* Default */

defaults_lookup(name, pairs)
        register char           *name;  /* Name to look up */
        register Defaults_pairs *pairs;       /* Default */
============================================================================
defaults_move()                      --NO LONGER SUPPORTED IN XView!!!!
============================================================================
defaults_remove()                    --NO LONGER SUPPORTED IN XView!!!!
============================================================================
defaults_remove_private()            --NO LONGER SUPPORTED IN XView!!!!
============================================================================
defaults_reread()                    --NO LONGER SUPPORTED IN XView!!!!
============================================================================
```

SunView/XView Interface **Comments**

```
============================================================================
defaults_set_character(path_name, value, status)
   char         *path_name;          /* Name to look up */
   char         value;               /* Character to set */
   int          *status;                /* Status flag */

defaults_set_character(resource, value)
        char    *resource;
        char    value;
============================================================================
defaults_set_enumeration()                    --NOT IN XView
Apparently not supported in XView....
it is in the defaults.h put no impl....
============================================================================
defaults_set_integer(path_name, value, status)
   char*path_name;                    /* Full node name */
   int value;                          /* Integer value */
   int *status;                          /* Status flag */
============================================================================
defaults_set_integer(resource, value)
        char    *resource;
        int             value;
============================================================================
defaults_set_string(path_name, value, status)
   char*path_name;                    /* Full node name */
   char*value;                       /* New string value */
   int *status;                          /* Status flag */

defaults_set_string(resource, value)
        char    *resource;
        char    *value;
============================================================================
defaults_set_prefix()         --NOT IN XView
============================================================================
defaults_special_mode()       --NOT IN XView
============================================================================
defaults_write_all()          --NOT IN XView
============================================================================
defaults_write_changed()      --NOT IN XView
============================================================================
defaults_write_differences()  --NOT IN XView
============================================================================
```

New XView Defaults

This section consists of an alphabetical listing of new XView defaults, sorted by package. Further information is available from the XView reference manual page.

Table A-4 New XView Defaults

`Alarm.Audible`	`term.alternateTtyswrc`
`Alarm.Visible`	`text.delimiterChars`
	`term.useAlternateTtyswrc`
`Font.Name`	
	`Text.AgainLimit`
`Icon.Font.Name`	`Text.AutoIndent`
`Icon.Footer`	`Text.AutoScrollBy`
`Icon.Header`	`Text.BlinkCaret`
`Icon.Pixmap`	`Text.CheckpointFrequency`
`Icon.X`	`Text.ConfirmOverwrite`
`Icon.Y`	`Text.DisplayControlChars`
	`Text.EnableScrollbar`
`keyboard.arrowKeys`	`Text.ExtrasMenuFilename`
`keyboard.leftHanded`	`Text.InsertMakesCaretVisible`
	`Text.LineBreak`
`Mouse.Multiclick.Space`	`Text.Margin.Bottom`
`Mouse.Multiclick.Timeout`	`Text.Margin.Left`
	`Text.Margin.Right`
`Notice.BeepCount`	`Text.Margin.Top`
`Notice.JumpCursor`	`Text.MaxFileSize`
`OpenWindows.dragRightDistance`	`Text.Retained`
`OpenWindows.multiClickTimeout`	`Text.StoreChangesFile`
`OpenWindows.WindowColor`	`Text.TabWidth`
`OpenWindows.DefaultName`	`Text.UndoLimit`
`OpenWindows.3DLook.Color`	
`OpenWindows.3DLook.Monochrome`	`window.mono.disableRetained`
`OpenWindows.SelectDisplaysMenu`	
	`Window.Color.Foreground`
`Scrollbar.LineInterval`	`Window.Color.Background`
`Scrollbar.PageInterval`	`Window.Columns`
`Scrollbar.RepeatDelay`	`Window.Footer`
	`Window.Header`
`Server.Name`	`Window.Height`
	`Window.Iconic`
`Term.BoldStyle`	`Window.InheritColor`
`Term.CheckpointFrequency`	`Window.Rows`
`Term.EnableEdit`	`Window.Scale`
`Term.InverseStyle`	`Window.Width`
`Term.MaxLogFileSize`	`Window.X`
`Term.UnderlineStyle`	`Window.Y`

CONVERT_TO_XDEFAULTS(1) USER COMMAND CONVERT_TO_XDEFAULTS(1)

NAME

convert_to_Xdefaults - converts a SunView1 defaults file to Xdefaults

SYNOPSIS

convert_to_Xdefaults *filename*

AVAILABILITY

Available with the OpenWindows Application Environment. For information about installing OpenWindows, refer to the OpenWindows Installation Manual.

DESCRIPTION

convert_to_Xdefaults is a shell script which uses sed(1) scripts to convert SunView1.x defaults to X Window (Xdefaults) defaults. convert_to_xview reads filename converting SunView1 defaults into their equivalent Xdefaults for XView. Defaults that are no longer supported or are not recognized as standard SunView1 defaults are commented out with an ! (exclamation-sign) at the beginning of the default entry. The output of conversion is directed to standard output (stdout). The defaults file should be located in your $HOME directory and should be named .Xdefaults

SAMPLES

A SunView1 defaults entries ...

```
/Text/Auto_indent              "True"
/Text/Extras_menu_filename     "/home/blinky/bob/.text_extras_menu"
/Scrollbar/Thickness           "20"
/Mail/Set/folder"              /home/blinky/bob/mail_folder"
/Text/Multi_click_timeout      "100"
```

is converted to the Xdefault...

```
Text.AutoIndent:               True
Text.ExtrasMenuFilename:          /home/blinky/bob/.text_extras_menu
!/Scrollbar/Thickness          "20"
!/Mail/Set/folder              "/home/blinky/bob/mail_folder"
!OpenWindows.MultiClickTimeout:
!(now in tenths of seconds rather than millisecs)  100
```

Note that the `/Scrollbar/Thickness` and `/Mail/Set/folder` entries were NOT converted but left in the file as comments. Whenever possible, instructions are included in the file for discrepancies between the two types of defaults. For instance, the comment "now in tenths of seconds" is useful information about the `OpenWindows.MultiClickTimeout` default. Comments and instructions should both be completely removed from the file.

FILES

`$OPENWINHOME/bin/xview/convert_to_Xdefaults` where `$OPENWINHOME` is the installation/ mount point for XView (/usr by default).

SEE ALSO

sunview(1), sed(1), textedit(1), vi(1)

B

Performance Hints

This appendix consists of some hints to help you fine-tune XView's performance.

Use Server Images

Use server images instead of memory pixrects. A server image is different from a memory pixrect in that the pixel values (the data bits) are stored on the X11 server, not on the client side. Thus, the pixel values do not have to be shipped over the wire each time they are drawn.

The following program demonstrates the use of `xv_rop()` with either memory pixrects or server images as the source for the operations.

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

/* create a memory pixrect containing the chesstool.icon bitmap */
static short chess_bits[] = {
#include <images/chesstool.icon>
};
mpr_static(chess_pixrect, 64, 64, 1, chess_bits);

Server_image    chess_server_image;

main()
{
    Frame       frame;
    Canvas      canvas;
    void        canvas_repaint_proc();
    frame = xv_create(0, FRAME,
       WIN_HEIGHT, 200,
       WIN_WIDTH,  200,
       0);

    /* create a server image containing the chesstool.icon bitmap */
    chess_server_image = xv_create(XV_NULL, SERVER_IMAGE,
       XV_HEIGHT,          64,
       XV_WIDTH,           64,
       SERVER_IMAGE_BITS,  chess_bits,
       0);
    canvas = xv_create(frame, CANVAS,
       CANVAS_REPAINT_PROC, canvas_repaint_proc,
       0);
    xv_main_loop(frame);
}
void
canvas_repaint_proc(canvas, paint_window, repaint_area)
   Canvas       canvas;
   Xv_window    paint_window;
   Rectlist     *repaint_area;
{
    xv_rop(paint_window, 10, 10, 64, 64, PIX_SRC, &chess_pixrect, 0, 0);
    xv_rop(paint_window, 75, 75, 64, 64, PIX_SRC, chess_server_image, 0, 0);

}
```

Create Resources Only When Needed

As a rule, create resources such as pop-ups only at the time they are required. For example, command and property pop-ups might not be used on every invocation of an application. Delaying the creation of such objects until they are actually requested helps to reduce resource utilization. Once such objects have been created they can simply be mapped and unmapped on demand, instead of having to be created and destroyed each time.

Canvas Paint Window

Be careful when retaining the canvas paint window. This increases the overall working set, especially on color where eight planes might be retained. Note also, that you are not guaranteed a returned window depending on server resource available. Retained is only a hint to the window manager to which it may or may not be adhered. You cannot rely on having the returned window available, and must always have a repaint process available to handle window damage.

Straight X Graphics

Use straight X graphics over the `pw_*` calls. They require more understanding of X and Xlib, but usually carry less overhead.

Unnecessary Syncing

Avoid unnecessary syncing with the server (`SERVER_SYNC`, `SERVER_SYNC_AND_PROCESS_EVENTS`) as each instance requires a reply from the server and thus can become expensive if overused.

If you plan to bypass `xv_main_loop()` by using `notify_start()`, or by using explicit dispatching via `notify_dispatch()`, be sure to sync with the server to insure all requests have been delivered and processed. Xlib and XView provide ways of doing this. The Xlib calls you can use are `XSync` or `XFlush`, for example:

Note: Xsynch and Xsynchronization can be very useful in debugging. See Appendix F for further details.

```
XSync((Display *) XV_DISPLAY_FROM_WINDOW(win),
    FALSE);
```

In XView, use the `attributes` `SERVER_SYNC` and `SERVER_SYNC_AND_PROCESS_EVENTS`, for example:

```
xv_set(XV_DISPLAY_FROM_WINDOW(window), FALSE,
    SERVER_SYNC, FALSE);
```

`SERVER_SYNC` is equivalent to `XSync`; `SERVER_SYNC_PROCESS_EVENTS` does an XSync and then insures that all the events are handled and dispatched to the appropriate client callbacks. Be careful not to overuse these synching mechanisms. Since all but the XFlush require a reply from the server before they will return, they can adversely affect performance. This can get very expensive if it is done too often.

An example of using the notifier is shown below.

$OPENWINHOME/share/src/sun/xview/examples/notifier/ntfy_do_dis.c

Miscellaneous

1) Use `xv_find` for FONT objects instead of `xv_create`. `xv_find` will use the font which already resides on the server, and create only when necessary.
2) Don't worry too much about clean-up on destruction since XView does most all of this for you. This includes destruction of sub-frames and especially when quitting the application.
3) Don't pre-create all windows/pop-ups/menus/images/etc in your startup code before calling `xv_main_loop()`. This results in your startup time being very slow. You can tune this later to cache the objects as they are needed, and/or move some of create calls back before `xv_main_loop()`.

C

Conversion Notes

This appendix contains supporting material for the shell script convert_to_xview, which you can use to perform either minimal or nearly full conversion of SunView applications to XView. In the section immediately below, those items which the script cannot convert automatically are flagged, and you can look them up in the index to complete the conversion. The section after that reprints the man page for this script. Here are some hints you may find useful:

Hints for Converting to XView

The shell script `convert_to_xview` places `/*XView CONVERSION*/` flags in your code wherever an in-line substitution cannot be made or additional attention is required. The table below lists possible conversion issues and references the appropriate sections in this manual. If you find conversion issues not covered by the automated conversion tool and/or the documentation, please send email to `windowbugs@sun.com.`

Table C-1 `convert_to_xview` *Cross Reference*

Flagged SunView1.x Reference	***Section***	***Comment***
#DEFINE_CURSOR_*	3.2	defunct
#DEFINE_ICON_FROM_IMAGE	3.2	defunct
CANVAS_MARGIN	4.2	defunct
CANVAS_PIXWIN	3.4	possibly defunct
CANVAS_RETAINED	3.4	only a hint
CURSOR_*	3.5	possibly defunct
emptysubwindow	3.15	defunct
esw_*	3.15	defunct

Flagged SunView1.x Reference	***Section***	***Comment***
FRAME_ARGC_PTR_ARGV	3.7	possibly defunct
FRAME_ARGS	3.7	possibly defunct
FRAME_EMBOLDEN_LABEL	4.2	defunct
FRAME_NTH_*	3.7	defunct, compat
FRAME_NTH_WINDOW	3.7	defunct
FRAME_SUBWINDOWS_ADJUSTABLE	4.2	defunct
fs_*	3.2	possibly defunct
fullscreen_*	3.2	possibly defunct
im_*	3.2	possibly defunct
input_read_event	3.14	possibly defunct
LINT_CAST	3.2	defunct (remove all references)
LOC_RGN*	3.2, 3.4	possibly defunct
MENU_BOXED	3.2	defunct
menu_display	3.9	defunct
MENU_FONT	3.1	defunct
MENU_SELECTED	4.2	defunct
MENU_STAY_UP	4.2	defunct
msgsw_*	3.15	defunct
panel_button_image	3.10	possibly defunct (use PANEL_LABEL_STRING)
PANEL_CU	3.10	defunct

Flagged SunView1.x Reference	***Section***	***Comment***
panel_get*	3.10	defunct, compat
PANEL_LABEL_IMAGE	3.10	possibly defunct (use PANEL_LABEL_STRING)
panel_make*	3.10	defunct
PANEL_MENU_*	3.10	defunct
panel_set*	3.10	defunct, compat
Pw_attribute_value	3.11	defunct
pw_close	3.4	defunct
pw_damaged	3.11	defunct
pw_dbl_*	3.11	defunct
PW_DBL_*	3.11	defunct
Pw_dbl_attribute	3.11	defunct
pw_donedamaged	3.11	defunct
pw_exposed	3.11	defunct
pw_open	3.11	defunct
pw_pfsysopen	3.6	defunct
pw_preparesurface	3.11	defunct
pw_region	3.11	defunct
pw_repairretained	3.11	defunct
pw_restrict*	3.11	defunct
pw_set_*	3.11	defunct
pw_traprop	3.11	defunct
pw_use_fast_mono	3.11	defunct
SCROLLBAR_*	3.12	possibly defunct
SCROLL_*	3.12	possibly defunct

Flagged SunView1.x Reference	***Section***	***Comment***
win_findintersect		defunct (use `win_pointer_under`)
win_getgfxwindow	3.14	defunct
win_setgfxwindow	3.14	defunct
struct cursor	3.2	defunct
struct icon	3.2	defunct
struct menu	3.9	defunct
struct pixwin	3.2	defunct
struct prompt	3.9	defunct
struct screen	3.2	defunct
struct tool	3.15	defunct
textsw_get*	3.13	possibly defunct
tool_	3.15	defunct
TOOL_*	3.15	defunct
tool_hs.h	3.2	remove
tool_parse_all	3.3, 3.15	defunct
tool_struct.h	3.2	remove
ttysw_*	3.13	possibly defunct
window_release_event_lock	3.14	defunct
window_set_cursor	3.1	defunct (use xv_set(window, WIN_CURSOR, foo))
WIN_CONSUME_	3.2	possibly defunct, compat, removed
win_enumall	3.14	defunct
win_enumerate_*	3.14	defunct
win_enumscreen	3.14	defunct
win_enum_input_device	3.14	defunct
win_errorhandler	3.14	defunct
WIN_FD	3.14t	defunct

Flagged SunView1.x Reference	***Section***	***Comment***
win_fdto	3.1	possibly defunct
win_getcursor	3.5	defunct
win_getowner	3.14	defunct
win_getparentwindow	3.14	defunct
win_getsavedrect	3.14	defunct
win_getscreenposition	3.14	defunct
win_getuserflag	3.14	defunct
win_get_button_order	3.14	defunct
win_get_designee		possibly defunct
win_get_event_timeout	3.14	defunct
win_get_fd	3.14	defunct
win_get_focus_event	3.14	defunct
win_get_kbd_mask	3.1	possibly defunct
win_get_pick_mask	3.14	possibly defunct
win_get_pixwin	3.14	defunct
win_get_scaling	3.14	defunct
win_get_swallow_event	3.14	defunct
win_get_tree_layer	3.14	defunct
win_initscreenfromargv	3.14	defunct
win_insertblanket	3.14	defunct
win_isblanket	3.14	defunct
win_is_input_device	3.14	defunct
WIN_KBD_INPUT		possibly defunct
win_numbertoname	3.1	possibly defunct
WIN_PICK_INPUT		defunct
win_release_event_lock	3.14	defunct

Flagged SunView1.x Reference	***Section***	***Comment***
win_removeblanket	3.14	defunct
win_remove_input_device	3.14	defunct
win_screendestroy	3.14	defunct
win_screenget	3.14	defunct
win_screennew	3.14	defunct
win_setcursor	3.15	defunct
win_setkbd	3.14	defunct
win_setms	3.14	defunct
win_setowner	3.14	defunct
win_setparentwindow	3.14	defunct
win_setsavedrect	3.14	defunct
win_setscreenposition	3.14	defunct
win_setuserflag	3.14	defunct
win_set_button_order	3.14	defunct
win_set_designee		possibly defunct
win_set_event_timeout	3.14	defunct
win_set_focus_event	3.14	defunct
win_set_input_device	3.14	defunct
win_set_kbd_mask	3.14	possibly defunct
win_set_pick_mask	3.14	possibly defunct
win_set_swallow_event	3.14	defunct
wmgr_*	3.14	possibly defunct

CONVERT_TO_XVIEW(1) USER COMMAND CONVERT_TO_XVIEW(1)

NAME

convert_to_xview — convert a SunView1 source program to XView source

SYNOPSIS

convert_to_xview [-m] *filename...*

AVAILABILITY

This command is available with the XView software distribution.

DESCRIPTION

`convert_to_xview` is a shell script which uses **sed**(1) scripts to convert SunView1.x programs to the XView Application Programming Interface (API). `convert_to_xview` parses *filename* and creates a new file with the XView API in the current directory called *`filename`*`.converted_to_xview`. The default conversion that is done is called FULL conversion. FULL conversion of SunView source converts everything to XView naming conventions regardless of API compatibility support (e.g., changes `WIN_FONT` to `XV_FONT` even though `WIN_FONT` would still work).

The other type of conversion is called MINIMAL conversion. MINIMAL conversion retains SunView compatibility wherever possible and inserts a unique flag and comments at every instance where manual conversion is necessary in C language source comment form. The flag and comments will look something like this:

```
#ifdef XVIEW_COMMENT
XView CONVERSION - Make sure to use xv_init to process the attrs
first. Sect 3.2
#endif
```

The original SunView1.x file is not modified. After the file is converted, you should then search for

```
XView CONVERSION
```

in the new converted program (filename.converted_to_xview). Use the conversion documentation, XView Version 2 Reference Manual: Converting SunView Applications, to determine the proper conversion for these flagged items. In some cases, the comments make references to sections in the manual.

OPTIONS

-m Perform minimal conversion only.

ENVIRONMENT

The script recognizes the environment variable $OPENWINHOME as the root directory for the installation point for convert_to_xview. By default it should be installed into the root directory '/'. Additionally, the **sed**(1) scripts that are used by convert_to_xview must be located in the $OPENWINHOME/conversion directory.

EXAMPLES

Convert foo.c from SunView1 to XView:

```
% convert_to_xview foo.c
----Converting File: foo.c
--Done
%
```

Now go in and edit (with your favorite text editor such as vi,textedit, etc.) the result of the conversion (my_program.c.converted_to_xview) and see if there is anythingthat didn't get converted:

```
    % textedit foo.c.converted
```

Do only minimal conversion of my_program.c & your_program.c to XView:

```
% convert_to_xview -m foo.c blah.c
----Converting File: foo.c
----Converting File: blah.c
--Done
%
```

The above would create two files new files and each will only had minimal conversion performed (just flags inserted).

FILES

$OPENWINHOME/bin/xview/convert_to_xview
$OPENWINHOME/bin/xview/convert_to_xview.README (this file)

Where $OPENWINHOME is the installation/mount point for XView.

SEE ALSO

sunview(1), sed(1), textedit(1), vi(1), sh(1)

D

Global Name Changes

Many functions have been renamed in XView to use reserved prefixes. This appendix contains a comprehensive listing of global name changes in both SunView 1 and XView.

Global Changes to Reserved Words

This section lists SunView and XView reserved words in tabular form.

Table D-1 *SunView vs. XView Reserved Words*

SunView 1.x	XView 1.0
sqroot	xv_sqroot
r	xv_random
anyof	xv_anyof
expand_name	xv_expand_name
short_to_char	pw_short_to_char
rank_to_selection	seln_rank_to_selection
substring	xv_substring
substrequal	xv_substrequal
everything	xv_everything
white_space	xv_white_space
blocking_wait	xv_blocking_wait
metanormalize	win_metanormalize
get_parent_dying	window_get_parent_dying
set_parent_dying	window_set_parent_dying
unset_parent_dying	window_unset_parent_dying
add_selection	panel_add_selection
button_init	panel_button_init

SunView 1.x	***XView 1.0***
message_init	panel_message_init
get_textsw_from_menu	textsw_from_menu
do_balance_beam	textsw_do_balance_beam
start_selection_tracking	textsw_start_seln_tracking
track_selection	textsw_track_selection
close_nonstd_fds_on_exec	textsw_close_nonstd_fds_on_exec
do_search_proc	textsw_do_search_proc
string_to_argv	textsw_string_to_argv
clean_up_move	textsw_clean_up_move
do_duplicate	textsw_do_duplicate
save_selection	textsw_save_selection
bold_mode	ttysw_bold_mode
nobold_mod	ttysw_nobold_mode
cim_clea	ttysw_cim_clear
cim_scroll	ttysw_cim_scroll
clear_mode	ttysw_clear_mode
deleteChar	ttysw_deleteChar
insertChar	ttysw_insertChar
insert_lines	ttysw_insert_lines
inverse_mode	ttysw_inverse_mode
noinverse_mode	ttysw_noinverse_mode
nounderscore_mode	ttysw_nounderscore_mode
underscore_mode	ttysw_underscore_mode
roll	ttysw_roll
swap	ttysw_swap
swapregions	ttysw_swapregions
vpos	ttysw_vpos
writePartialLin	ttysw_writePartialLine
imagerepair	ttysw_imagerepair
blinkscreen	ttysw_blinkscreen
drawCursor	ttysw_drawCursor
fixup_display_mode	ttysw_fixup_display_mode
pclearscreen	ttysw_pclearscreen
copyline	ttysw_pcopyline
copyscreen	ttysw_pcopyscreen

SunView 1.x	***XView 1.0***
displayscreen	ttysw_pdisplayscreen
prepair	ttysw_prepair
pselectionhilit	ttysw_pselectionhilite
pstring	ttysw_pstring
removeCursor	ttysw_removeCursor
restoreCursor	ttysw_restoreCursor
saveCursor	ttysw_saveCursor
screencomp	ttysw_screencomp
ansi_escape	ttysw_ansi_escape
ansiinit	ttysw_ansiinit
constructargs	wmgr_constructargs
initrandom	xv_initrandom
free_filter_table	xv_free_filter_table
parse_filter_table	xv_parse_filter_table
skip_over	xv_skip_over
getlogindir	xv_getlogindir
x_input_readevent	xview_x_input_readevent
ansi_string	ttysw_ansi_string
pclearline	ttysw_pclearline
destroy_font_struct	font_destroy_struct
file_input_stream	xv_file_input_stream
file_input_stream_close	xv_file_input_stream_close
file_input_stream_getc	xv_file_input_stream_getc
file_input_stream_get_pos	xv_file_input_stream_get_pos
ile_input_stream_set_pos	xv_file_input_stream_set_pos
file_input_stream_ungetc	xv_file_input_stream_ungetc
file_input_stream_fgets	xv_file_input_stream_fgets
file_input_stream_chars_avail	xv_file_input_stream_chars_avail
file_input_stream_data	xv_file_input_stream_data
file_input_stream_ops	xv_file_input_stream_ops
file_output_stream	xv_file_output_stream
file_output_stream_close	xv_file_output_stream_close
file_output_stream_flush	xv_file_output_stream_flush
file_output_stream_put	xv_file_output_stream_putc
file_output_stream_fputs	xv_file_output_stream_fputs

SunView 1.x	***XView 1.0***
`file_output_stream_get_pos`	`xv_file_output_stream_get_pos`
`file_output_stream_data`	`xv_file_output_stream_data`
`file_output_stream_ops`	`xv_file_output_stream_ops`
`filter_comments_stream`	`xv_filter_comments_stream`
`filter_comments_stream_clos`	`xv_filter_comments_stream_close`
`filter_comments_stream_get`	`xv_filter_comments_stream_getc`
`filter_comments_stream_get_pos`	`xv_filter_comments_stream_get_pos`
`filter_comments_stream_ungetc`	`xv_filter_comments_stream_ungetc`
`filter_comments_stream_chars_avail`	`xv_filter_comments_stream_chars_avail`
`filter_comments_stream_data`	`xv_filter_comments_stream_data`
`filter_comments_stream_ops`	`xv_filter_comments_stream_ops`
`make_0list`	`xv_make0list`
`make_1list`	`xv_make1list`
`make_insert_visible`	`textsw_make_insert_visible`
`view_from_panel_item`	`text_view_frm_p_itm`
`pos`	`ttysw_pos`
`do_move`	`xv_do_move`

E

Converting Graphics from Pixwin to Xlib

Goal

This appendix describes how to convert Pixwin graphics function calls into Xlib graphics function calls. This conversion will allow the resulting converted application to run with improved performance and allow their application to be much more portable (and/or displayable) to other X11 environments. Without this complete conversion, your program may not look the way you anticipated on all X servers.

Background

In SunView applications, the standard graphics interface was called Pixwin. Pixwin is a logical windowing interface to the raster-op style Pixrect graphics interface. The Pixwin and Xlib graphics interfaces are very similar and nearly any graphics written for Pixrect can be easily rewritten to Xlib graphics interfaces (and vice-versa).

How to Use This Guide

This guide provides a simple function-by-function conversion recipe. In order to use this guide, first find the `pw_*()` function that you wish to convert (functions are in alphabetical order), then examine the recipe. Each function will include one or more Xlib equivalents, example code showing usage of the Xlib function to perform the same graphics operation, notes on function-specific arguments and descriptions of other Xlib functions necessary to make use of the Xlib graphics function.

At the end of this appendix, there is a section describing the creation of X Graphics Contexts for your converted graphics. You may need to consult this section as a result of converting many of your Pixwin function calls.

Note that this appendix assumes the reader has complete Pixwin graphics and Xlib graphics interface documentation. Please be sure to have these documents available to consult when you have any questions about Pixwin or Xlib graphics.

When converting your Pixwin/Pixrect graphics operations to Xlib, you should be very careful to not make any assumptions about the foreground and background pixel values. In SunView they were logical ’0’ and ’1’. In the X Window System, this is not the case and many different X servers use different values that would cause serious display problem if this principle were violated. For your convenience, Xlib provides macros for getting at the logical "black" and "white" pixel values.

```
foreground = BlackPixel(dpy, scr);
background = WhitePixel(dpy, scr);
```

Functions

This section describes the `pw_*()` functions to be converted. Each function is listed below in alphabetical order.

pw_char(pw, x, y, op, fontname, character)

Use the XLib call `XDrawImageString()` after converting the character into a string. See the section on `pw_text()` for more details.

pw_get(pw, x, y)

Approximate Xlib Equivalent:

```
XGetImage(display, drawable, x, y, 1, 1, planemask,
format);
XGetPixel(image, 0, 0);
```

Example Code:

```
Display *display;
Drawable drawable;
XImage *image;
int value;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Get image using XGetImage() */
image = XGetImage(display, drawable, x, y, 1, 1,
AllPlanes, ZPixmap);

/* Get pixel value from image using XGetPixel() */
value = (int)XGetPixel(image, 0, 0);
```

Functions used:

```
XGetImage(display, drawable, x, y, width, height,
plane_mask, format);
```

`display, drawable`	See example code.
`x, y`	Same as pw_get call.
`width, height`	Should be 1.
`format`	(1) XYPixmap: gets only the bit planes specified in the `plane_mask` argument. (2) ZPixmap : Sets to 0 the bits in all planes not specified in the `plane_mask` argument. No range checking is done.
`plane_mask` -	(1) AllPlanes. (2) or you can specify the planes, you want to be returned.

```
XGetPixel(image, x, y);
```

`image`	Returned from `XGetImage` call
`x, y`	Should be 0's.

`pw_line(pw, x0, y0, x1, y1, brush, tex, op);`

Approximate Xlib Equivalent:

```
XDrawLine(display, drawable, gc, x0, y0, x1, y1);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;

/* Get display and drawable. */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Create a graphics context /
gc_val.foreground = value;
gc_val.function = op;
gc_val.line_width = brush->width;
gc_val.line_style = LineSolid;
gc = XCreateGC(display, drawable,
GCForeground|GCFunction|GCLineWidth|GCLineStyle,
&gc_val);

/* Used the graphics context to draw the line */
XDrawLine(display, drawable, gc, x0, y0, x1, y1);
```

Notes:

`XDrawLine()` uses the foreground pixel and function attributes of the graphics context to draw the line.

Refer to section on graphics contexts for mapping `pw ops` to `gc_val.functions`.

SunView attribute	->	Graphics context attribute
`op`		`gc.function`
`brush->width`		`gc_val.line_width`
`tex->pattern`		`gc_val.line_style`
`NULL`		`LineSolid`
`pw_tex_dashed`		`LineOnOffDash`
`tex->offset`		`gc_val.dash_offset`

`pw_tex_dashdot`
`pw_tex_dashdotdotted`
`pw_tex_longdashed`
		For these 3 pw attributes, create an array of chars of the list of penup the list of penup and pendowns and use `gc_val.dashes` on your GC.

* If you don’t care about line width, use `line_width` of 0 which is faster.

For more information see section on graphics contexts at the end of this appendix.

Functions used:

```
XDrawLine(display, drawable, gc, x, y);
```

`display, drawable -`	See example code.
`gc -`	Graphics context to use.
`x, y -`	Coordinate to draw point.

`pw_polygon_2 (pw, dx, dy, nbds, npts, vlist, op, spr, sx, sy);`

Xlib Equivalent:

```
FillPolygon(display, drawable, gc, points, npoints,
shape, mode);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;
XPoint *points;
insigned long valuemask = 0;
int i;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/*
 * Allocate memory for your XPoint struct.
 * Initialise Xpoint struct with the corresponding
 * fields from vtlist.
 * remember Xlib uses shorts instead of integers,
 * and you need to add the offset dx or dy
 */
points = (XPoint *)malloc(sizeof(short) * 2 * npts);
for (i = 0; i < npts; ++i) {
points[i].x = (short) vtlist[i].x + dx;
points[i].y = (short) vtlist[i].y + dy;
}

/* Create a graphics context */
gc_val.function = op;
valuemask |= GCFunction;
if (spr == NULL) {
/* fill polygon with a solid color */
gc_val.foreground = whatever_fg_color_you_desire;
valuemask |= GCForeground;
gc_val.fill_style = FillSolid;
valuemask |= GCFillStyle;

} else {/* spr == Server Image */
/*
 * if it is not see the ’notes’ section below on
 * how to convert from a pixrect to a Server Image
      */


int src_depth = (int)xv_get(spr, WIN_DEPTH);
int dst_depth = (int)xv_get(pw, WIN_DEPTH);

if (scr_depth == dst_depth) {
gc_val.tile = xv_get(spr, XV_XID);
valuemask |= GCTile;
gc_val.fill_style = FillTiled;
} else{
gc_val.stipple = xv_get(spr, XV_XID);
valuemask |= GCStipple;
gc_val.fill_style = FillOpaqueStipple;
}

gc_val.ts_x_origin = dx - sx;
gc_val.ts_y_origin = dy - sy;

valuemask |=
(GCFillStype|CGTileStipXOrigin|CGTileStipYOrigin);
}

gc = XCreateGC(display, drawable, valuemask,
&gc_val);

/* draw a filled polygon */
XFillPolygon(display, drawable, gc,
points, npts, Complex, CoordModeOrigin);
```

If there are many polygons to be drawn, split the points and use a "for" loop with the above call.

Notes:

Refer to section on graphics contexts for mapping `pw ops` to `gc_val.functions`.

If the `spr` is a pixrect and not a Server Image, you can convert it as follows:

```
Server_image tile;
tile = xv_create(NULL,
SERVER_IMAGE,
XV_HEIGHT,spr->pr_height,
XV_WIDTH,spr->pr_width,
SERVER_IMAGE_DEPTH,spr->pr_depth,
SERVER_IMAGE_BITS,(mpr_d(spr))->md_image,0);
```

** Refer to the section on XFillPolygon() in any Xbook for more details.

<u>Functions used:</u>

```
XFillPolygon(display, drawable, gc, points, npoints,
shape, mode);
```

`display, drawable -`	See example code.
`gc -`	Graphics context to use.
`points -`	Pointer to an array of vertices.
`npoints -`	# of points in the array.
`shape -`	A hint to help the server improve preformance.
`. Complex`	(slowest)
`. Nonconvex`	
`. Convex`	(fastest)
`mode -`	Should be `CoordModeOrigin`.

`pw_polyline(pw, dx, dy, npts, ptlist, mvlist, brush, tex, op);`

<u>Approximate Xlib Equivalent:</u>

```
XDrawLines(display, drawable, gc, points, npoints,
   mode);
```

<u>Example Code:</u>

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;
XPoint *points;
```

```
/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/*
 * Load the point list with npts coordinates.
 * remember Xlib uses shorts instead of integers,
 * and you need to add the offset dx or dy
 */
points = (XPoint *)malloc(sizeof(short) * 2 * npts);
for (i = 0; i < npts; ++i) {
points[i].x = (short) ptlist[i].x + dx;
points[i].y = (short) ptlist[i].y + dy;
}

/* Create a graphics context */
gc_val.foreground = value;
gc_val.function = op;
gc_val.line_width = brush->width;
gc_val.line_style = LineSolid;
gc = XCreateGC(display, drawable,
GCForeground|GCFunction|GCLineWidth|GCLineStyle,
&gc_val);

XDrawLines(display, drawable, gc, points, npts,
CoordModeOrigin);
```

Notes:

Refer to section on graphics contexts for mapping `pw ops` to `gc_val.functions`.

Use brush and tex to set `gc_val.line_width`, `gc_val.line_style`, `gc_val.dashes`, and `gc_val.dash_offset`. See `pw_line` for more details.

If `mvlist` equals `POLY_CLOSE` set the last and first points the same in `XDrawLines()`.

There are lot of things which can be done efficiently if the developer reads the section of any X book on `XDrawLines()`.

Functions used:

```
XDrawLines(display, drawable, gc, points, npoints,
mode);
```

`display, drawable -`	See example code.
`gc -`	Graphics context to use.
`points -`	Pointer to an array of points.
`npoints -`	# of points in the array.
`mode -`	Should be `CoordModeOrigin`.

pw_polypoint(pw, dx, dy, npts, ptlist, op); *

Approximate Xlib Equivalent:

```
XDrawPoints(display, drawable, gc, points, npoints,
mode);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;
XPoint *points;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/*
 * Load the point list with npts coordinates.
 * remember Xlib uses shorts instead of integers,
 * and you need to add the offset dx or dy
 */
points = (XPoint *)malloc(sizeof(short) * 2 * npts);
for (i = 0; i < npts; ++i) {
points[i].x = (short) ptlist[i].x + dx;
points[i].y = (short) ptlist[i].y + dy;
}

/* Create a graphics context */
gc_val.function = op;
gc_val.foreground = foreground_value;
gc = XCreateGC(display, drawable, GCFunction,
&gc_val);

XDrawPoints(display, drawable, gc, points, npts,
CoordModeOrigin);
```

Notes:

Refer to section on graphics contexts for mapping `pw ops` to `gc_val.functions`.

`gc_val.foreground` should be set appropriately.

See graphics contexts for more details.

Functions used:

```
XDrawPoints(display, drawable, gc, points, npoints,
mode);
```

`display, drawable -`	See example code.
`gc -`	Graphics context to use.
`points -`	Pointer to an array of points.
`npoints -`	# of points in the array.
`mode -`	Should be `CoordModeOrigin`.

pw_put(pw, x, y, value)

Approximate Xlib Equivalent:

```
XDrawPoint(display, drawable, gc, x, y);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;

/* Get display and drawable. */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Create a graphics context */
gc_val.foreground = value;
gc = XCreateGC(display, drawable, GCForeground,
&gc_val);

/* Used the graphics context to draw the point. */
XDrawPoint(display, drawable, gc, x, y);
```

Notes:

`XDrawPoint()` uses the foreground pixel and function attributes of the graphics context to draw the point. For more information see section on graphics contexts.

Functions used:

```
XDrawPoint(display, drawable, gc, x, y);
display, drawable -See example code.
gc -Graphics context to use.
x, y -coordinate to draw point.
```

pw_read(pr, dx, dy, dw, dh, op, pw, sx, sy)

Approximate Xlib Equivalent:

```
XGetImage(display, drax, sy, dw, dh, planemask,
format);
```

Example Code:

```
Display *display;
Drawable drawable;
XImage *image;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Get image using XGetImage() */
image = XGetImage(display, drawable,
sx, sy, dw, dh, AllPlanes, ZPixmap);
```

Notes:

To convert a XImage struct to a memory Pixrect:

```
image_mpr.pr_depth = image->depth;
image_mpr.pr_height = image->height;
image_mpr.pr_width = image->width;
image_mpr.pr_data = (caddr_t) (&image_mpr_data);
image_mpr_data.md_linebytes = image->bytes_per_line;
image_mpr_data.md_image = (short *) image->data;
image_mpr_data.md_offset.x = 0;
image_mpr_data.md_offset.y = 0;
image_mpr_data.md_primary = 0;
image_mpr_data.md_flags = 0;
```

Functions used:

```
XGetImage(display, drawable, x, y, width, height,
plane_mask, format);
```

`display,drawable -`	See example code.
`x, y -`	Same as pw_read call.
`width, height -`	Same as pw_read call.
`format -`	(1) XYPixmap: gets only the bit planes specified in the plane_mask argument. (2) ZPixmap: Sets to 0 the bits in all planes not specified in the plane_mask argument. No range checking is done.
`plane_mask -`	(1) AllPlanes. (2) or you can specify the planes, you want to be returned.

`pw_rop(pw, dx, dy, dw, dh, op, pr, sx, sy)`

Approximate Xlib Equivalent:

```
XFillRectangle(display, drawable, gc, dx, dy, w, h);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Create a graphics context,
 * this example will only handle NULL src pr’s
 * for non-NULL src pr’s see below
 */
```

```
gc_val.foreground = foreground_color;
gc_val.function = op;
gc = XCreateGC(display, drawable,
GCForeground|GCFunction, &gc_val);

/* Used the graphics context to do the rop */
XFillRectangle(display, drawable, gc, dx, dy, dw,
dh);
```

<u>Notes:</u>

Refer to section on graphics contexts for mapping `pw ops` to `gc_val.functions`. For more information see section on graphics contexts.

Get the `src_info` using
`DRAWABLE_INFO_MACRO(src,src_info);` where
`src = (Xv_Opaque) pr;`

The source `pr` can be NULL, a Server Image, a Client Pixrect, or a Window. The case of a NULL pr is handled in the above example. The other cases are handled as follows.

(1) pr == Server Image

```
   int src_depth,dest_depth;

   gc_val.ts_x_origin = x;
   gc_val.ts_y_origin = y;
```

Compare the depths of the destination display and source server image using `xv_depth(..)`.

```
if (dest_depth == 1 && src_depth == 1) {
x_display = XV_SERVER_FROM_WINDOW(my_xview_window);
x_screen = XV_SCREEN_FROM_WINDOW(my_xview_window);
gc_val.foreground = BlackPixel(x_display, x_screen);
gc_val.background = WhitePixel(x_display, x_screen);
   gc_val.stipple = xv_get(pr, XV_XID);
   gc_val.fill_style = FillOpaqueStippled;
   gc = XCreateGC(display, drawable, flags,
      &gc_val);
```

Remember to set the flags correctly. Refer to section on GCs.

```
} else if (dest_depth == 8 && src_depth == 1) {
   gc_val.stipple = xv_get(pr, XV_XID);
   gc_val.fill_style = FillOpaqueStippled;
   gc = XCreateGC(display, drawable, flags, &gc_val);
```

Remember to set the flags correctly. Refer to section on GCs.

```
   } else if (dest_depth == src_depth) {
   gc_val.tile = xv_get(pr, XV_XID);
   gc_val.fill_style = FillTile;
   gc = XCreateGC(display, drawable, flags, &gc_val);
```

Remember to set the flags correctly. Refer to section on GCs.

```
      }
   XFillRectangle(display, drawable, gc, dx, dy, dw,
dh);
```

(2) pr == Window

Here the destination display depth and the source Window depth need to be equal.

```
XCopyArea(display, (Drawable)xv_get(pr,
XV_XID),drawable, gc, sx, sy, dw, dh, dx, dy);
```

(3) pr == client Pixrect;

This is what needs to be done.

a) Create the image using `XCreateImage()`.
b) If the image is a color image the pixel value (indexes) must be corrected to reflect the way colormap segments are handled in X. Consult the XView documentation regarding colormaps.
c) Draw the `Ximage` to the screen using `XPutImage()`.

pw_stencil(dpw, dx, dy, dw, dh, op, stpr, stx, sty, spr, sx, sy)

Approximate Xlib Equivalent:

```
XFillRectangle(display, drawable, gc, dx, dy, dw,
dh);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;

/* Get display and drawable */
display=(Display *)xv_get(dpw, XV_DISPLAY);
drawable=(Drawable)xv_get(dpw, XV_XID);

/* Create a graphics context */
/* set the fg color, bg color and function
appropriately */
gc_val.function = op;
gc_val.foreground = foreground_color;
gc_val.background = background_color;
/* set the clip_mask, here we are assuming spr is a
1 bit
 * Pixmap.  If it is not you will need to convert it.
 */
gc_val.clip_mask = stpr;

gc = XCreateGC(display, drawable,
GCFunction|GCForeground|GCBackground|GCClipMask,
&gc_val);

if (spr) {
/* do approximately the same thing as pw_rop() */
} else {
/* Used the graphics context to do the stencil */
XFillRectangle(display, drawable, gc, dx, dy, dw,
dh);
}
```

Notes:

The Xlib code for `pw_stencil` is basically the same code `aspw_rop` with a clipmask add to the graphics context.

This example assumes that the stencil (`stpr`) is a 1 bit Pixmap. If that is not the case `stpr` needs to be converted into a Pixmap. Pixmaps are created using `XCreatePixmap()`. See any X manual for documentation on how this is done. Once you have this the Pixmap can use it in your `XCreateGC()`.

Refer to section on graphics contexts for mapping `pw ops` to gc_val.functions.

Functions used:

```
XFillRectangle(display, drawable, gc, dx, dy, w, h);
```

`display, drawable -`	See example code.
`gc -`	Graphics context to use.
`dx, dy -`	Upper-left hand corner of the rectangle. this should be dx and dy.
`w, h -`	Dimensions of the rectangle to be filled. This should be dw and dh.

`pw_text(pw, x, y, op, font, string)`
`pw_ttext(pw, x, y, op, font, string)`

Approximate Xlib Equivalent:

```
pw_text =>  XDrawImageString(display, drawable, gc,
x, y, string, length);
pw_ttext => XDrawString(display, drawable, gc, x, y,
string, length);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;
XFontStruct *font;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* load and query font */
font = XLoadQueryFont(display, fontname);

/* Create a graphics context */
gc_val.font  = font->fid;
gc_val.foreground = foreground_value;
gc_val.background = background_value;
gc_val.function = op;

gc = XCreateGC(display, drawable,
GCFunction|GCForeground|GCBackground, &gc_val);
```

```
XDrawString(display, drawable, gc, x, y, string,
strlen(string));
/*
XDrawImageString(display, drawable, gc, x, y, string,
strlen(string));
 */
```

Notes:

`XDrawImageString()` draws non-transparent text using both foreground and background attributes in the graphics context.

`XDrawString()` draws transparent text using only the foreground attribute in the graphics context. It does not affect any other pixels in the bounding box.

Use `XLoadQueryFont(display,fontname)` to load a font instead of `pf_open(fontname)`. There may not be a one to one correspondence between your SunView font name and a X font name. Use the utilities `xlsfonts` and `xfd` to select a new X font.

`gc_val.foreground` and `gc_val.background` should be set to the respective pixel values.

Refer to section on graphics contexts for mapping `pw ops` to `gc_val.functions`.

For more information see section on graphics contexts.

Functions used:

```
XDrawString(display, drawable, gc, x, y, string,
length);
```

`display, drawable` -	See example code.
`gc` -	Graphics context to use.
`x, y` -	Starting position of string.
`string` -	The character string.
`length` -	Size of the character string.

`pw_vector(pw, x0, y0, x1, y1, op, value)`

Approximate Xlib Equivalent:

```
XDrawLine(display, drawable, gc, x0, y0, x1, y1);
```

Example Code:

```
Display *display;
Drawable drawable;
GC gc;
XGCValues gc_val;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Create a graphics context */
gc_val.foreground = value;
gc_val.function = op;

gc = XCreateGC(display, drawable,
GCFunction|GCForeground, &gc_val);

XDrawLine(display, drawable, gc, x0, y0, x1, y1);
```

Notes:

`XDrawLine()` uses the foreground pixel and function attributes of the graphics context to draw the line. Refer to section on graphics contexts for mapping pw ops to `gc_val.functions`.

Functions used:

```
XDrawLine(display, drawable, gc, x, y);
```

`display, drawable`	See example code.
`gc -`	Graphics context to use.
`x, y -`	Coordinate to draw point.

`pw_writebackground(pw, dx, dy, w, h, op)`

Approximate Xlib Equivalent:

```
XFillRectangle(display, drawable, gc, dx, dy, w, h);
```

Example Code:

```
Display *display;
```

```
Drawable drawable;
GC gc;
XGCValues gc_val;

/* Get display and drawable */
display=(Display *)xv_get(pw, XV_DISPLAY);
drawable=(Drawable)xv_get(pw, XV_XID);

/* Create a graphics context */
/* set foreground pixel to actual background pixel
value */
gc_val.foreground = background;
gc = XCreateGC(display, drawable, GCForeground,
&gc_val);

/* Used the graphics context to paint the background
*/
XFillRectangle(display, drawable, gc, dx, dy, w, h);
```

<u>Notes:</u>

In this example `XFillRectangle()` use the foreground pixel and function attributes of the graphics context to paint the background.

The function attribute of the graphics context should be set according to the SunView op required. `GXcopy` is the default function.

The foreground and background attributes should be set appropriately.

See graphics contexts for more details.

<u>Functions used:</u>

```
XFillRectangle(display, drawable, gc, dx, dy, w, h);
```

`display, drawable` -	See example code.
`gc` -	Graphics context to use.
`dx, dy` -	Upper-left hand corner of the rectangle
`w, h` -	Dimensions of the rectangle to be filled.

Graphics Contexts

All Xlib graphics routines use graphics contexts to draw images. Graphics contexts can be created, changed, used and reused. Below is the Pixwin ops to logical functions (`gc_val.function`) mapping.

Pixwin	Xlib
PIX_CLR	= GXclear
PIX_SET	= GXset
PIX_DST	= GXnoop
PIX_NOT(PIX_DST)	= GXinvert
PIX_SRC	= GXcopy
PIX_NOT(PIX_SRC)	= GXcopyInverted
PIX_SRC & PIX_DST)	= GXand
PIX_SRC & PIX_NOT(PIX_DST))	= GXandReverse
PIX_NOT(PIX_SRC) & PIX_DST)	= GXandInverted
PIX_SRC ^ PIX_DST)	= GXxor
PIX_SRC | PIX_DST)	= GXor
PIX_NOT(PIX_SRC) & PIX_NOT(PIX_DST))	= GXnor
PIX_NOT(PIX_SRC) ^ PIX_DST)	= GXequiv
PIX_SRC | PIX_NOT(PIX_DST))	= GXorReverse
PIX_NOT(PIX_SRC) | PIX_DST)	= GXorInverted
PIX_NOT(PIX_SRC) | PIX_NOT(PIX_DST))	= GXnand

Example of creating a graphics context.

```
GC gc;
XGCValues gc_val;
Display x_display;
Screen  x_screen;

x_display = XV_SERVER_FROM_WINDOW(my_xview_window);
x_screen = XV_SCREEN_FROM_WINDOW(my_xview_window);
gc_val.foreground = BlackPixel(x_display, x_screen);
gc_val.background = WhitePixel(x_display, x_screen);
gc = XCreateGC(x_display, drawable,
GCForeground|GCBackground, &gc_val);
```

Example of changing an existing graphics context.

```
XGCValues gc_val;
Display x_display;
Screen  x_screen;

x_display = XV_SERVER_FROM_WINDOW(my_xview_window);
x_screen = XV_SCREEN_FROM_WINDOW(my_xview_window);
gc_val.foreground = BlackPixel(x_display, x_screen);
XChangeGC(x_display, gc, GCForeground, &gc_val);
```

<u>Functions used:</u>

```
XCreateGC(display, drawable, valuemask, values);
```

`display -`	Specifies the display.
`drawable -`	Specifies the drawable.
`valuemask -`	Attributes of the gc you want to change.
`value -`	Pointer to the XGCValues structure.

F

Debugging Converted Programs

This appendix lists suggestions for debugging your XView program. Further debugging information can be found in the manual describing dbxtool, as well as the Xlib Programming Manual.

Using XSync() to Debug Programs

Because `XSync()` forces every request to be flushed to the server as it happens, it can be useful debugging XView programs. By default, requests get queued up, and you'll often receive an error quite a while after it really occurred. By sprinkling `XSync()` calls in the area where you suspect a bug, requests can be flushed as they happen.

For example, suppose if you were running the xfish program (a common X demo program), and you got the following error message:

```
X Error:  BadMatch, invalid parameter attributes
Request Major code 1 ()
Request Minor code
ResourceID 0x500000
Error Serial #3
Current Serial #36
```

Because the error said the request code was 1, you would consult `Xproto.h` and see that its a CreateWindow call and add the `XSync()` after it.

Refer to Xsynch and Xsynchronize in the Xlib Programming and Reference Manuals.

Disabling XGrabs in the Fullscreen Package

Typically, when you step through code that uses the Fullscreen package, you get hung up at the point where the code grabs the server, keyboard, or pointer. Four global variables in the Fullscreen package can be used to disable `XGrabs`. These variables can be set in dbx or in the source. They should be set at the very beginning of the application. For example, cmdtool can be executed in fullscreen debug mode in any of the following three ways: by typing `cmdtool -Wfsdb` or `cmdtool -fullscreendebug` on the command line; by adding the line `fullscreen.debug:True` to the `.Xdefaults` file, or by making the line `fullscreendebug=1;be` the first line in the function `main()` (be sure to `#include <xview/fullscreen.h>).`

Likewise, you can set these variables in a debugger after setting a breakpoint in `main()`. If the fullscreen variables are used at places other than at the start of the application, the screen may be locked up.

Note that adding the line `fullscreen.debug:True` to your `.Xdefaults file` will make all XView applications run in fullscreen debug mode:

Variable	***Comments***
`fullscreendebugserver`	if non-zero server grabs are disabled
`fullscreendebugkbd`	if non-zero keyboard grabs are disabled
`fullscreendebugptr`	if non-zero pointer grabs are disabled (0,ICON,...)
`fullscreendebug`	if non-zero server/keyboard/pointer grabs are disabled

Note: The default for all the above is zero.

In addition, there are also four command line options that can be used for these same purposes. The command line options are:

SHORT	LONG	X RESOURCE NAME
`-Wfsdb`	-fullscreendebug	fullscreen.debug (True/False)
`-Wsdbs`	-fullscreendebugserver	fullscreen.debugserver
`-Wfsdbk`	-fullscreendebugkbd	fullscreen.debugkbd
`-Wfsdbp`	-fullscreendebugptr	fullscreen.debugptr

An example dbx session is shown below. The first session is done without using the debug variables and getting stuck. The second session shows the same session using the fullscreen variables.

First session (without debug variables)

```
#Run debugger

% dbx fullscreen
Reading symbolic information...
Read 4091 symbols

#Set breakpoint in callback function
(dbx) stop in grab
(2) stop in grab

#Run executable
(dbx) r
Running: fullscreen
<Click select "Fullscreen" button>

#Debugger stops at breakpoint
stopped in grab at line 46
   46      Panel      panel = (Panel)xv_get(item, PANEL_PARENT_PANEL);
#Set another breakpoint which is in the middle of fullscreen code
(dbx) stop at 69
(4) stop at 69

#Continue execution and....
(dbx) c

<Click select button>

#...we stop at the breakpoint but we can’t do anything since the
#XGrabs performed during fullscreen creation blocks input to ALL other
# windows

stopped in grab at line 69
   69              break;
#At this point the dbx process has to be killed from another terminal
```

In this second example, the same session is shown using fullscreen debug variables.

```
#Run debugger

% dbx fullscreen
Reading symbolic information...
Read 4091 symbols
(dbx)

#Set breakpoint in callback function
(dbx) stop in grab
(2) stop in grab

#Run executable with fullscreen debug flag turned on
(dbx) r -Wfsdb
Running: fullscreen -Wfsdb
<Click select "Fullscreen" button>
#Debugger stops at breakpoint
stopped in grab at line 46
   46      Panel      panel = (Panel)xv_get(item, PANEL_PARENT_PANEL);
#Set another breakpoint which is in the middle of fullscreen code
(dbx) stop at 69
(4) stop at 69
#Continue execution and....
(dbx) c
<Click select button in application window>

#...we stop at the breakpoint and still have control off the pointer/
keyboard
stopped in grab at line 69
   69               break;

#Print event struct
(dbx) p *event
*event = {
        ie_code      = 32563
        ie_flags     = 0
        ie_shiftmask = 0
        ie_locx      = 73
        ie_locy      = 27
        ie_time      = {
               tv_sec  = 26952
                tv_usec = 820000
         }
```

Second session (with debug variables) continued:

```
        action       = 31799
        ie_win       = 966428
        ie_string    = (nil)
        ie_xevent    = 0xdffff42c
}

#Continue stepping through code...
(dbx) n
stopped in grab at line 72
   72       xv_destroy(fs);
(dbx) n
stopped in grab at line 75
   75       printf("event was button %d\n", event_id(event) - BUT_FIRST
+ 1);
(dbx) n
event was button 1
stopped in grab at line 77
   77   }
(dbx) q
%
```

Index