CHAPTER 8

# Other Standard Clients

This chapter gives an overview of other clients available with X, including window and display information clients, printing utilities, the *xkill* program, and several "desk accessories." It also examines some of the features common to applications written with the X Toolkit.

The standard X11 distribution includes many useful clients. Most of these are included in OpenWindows Release Three. We have grouped these X11 clients according to basic functionality:

- Desk accessories: *xclock*, *oclock*, *xcalc*, *xbiff*, *xload*, and *xman*.
- Text editor: *xedit*.
- Window and display information programs: *xwininfo*, *xlswins*, *xlsclients*, *xdpyinfo*, and *psps*.
- Programs to remove a client window: *xkill*, pam.

In many cases we refer you to OPEN LOOK or OpenWindows alternatives described elsewhere in the book. Most sections in this chapter are intended to acquaint you with the major features of some of the available clients. Additional detailed information is provided on the reference pages for each client in Part Three of this guide.

Many of the standard clients ere written using a programming library called the X Toolkit. As explained in Chapter 2, the X Toolkit includes a set of predefined components (or widgets), known as the Athena widget set. And the OPEN LOOK Intrinsics Toolkit includes a different set of widgets. Widgets make it easier to create complex applications; they also ensure a consistent user interface between applications. In discussing various clients in this chapter, we'll point out some of the features attributable to the X Toolkit.

For a comprehensive treatment of the X Toolkit, software developers should refer to Volume Four, *X Toolkit Intrinsics Programming Manual*, and Volume Five, *X Toolkit Intrinsics Reference Manual*.

You can start these clients from the command line in any *xterm* window or, if you like, you can add them to your window manager's menu (see Chapter 13, *Customizing olwm*).

You can terminate any of these clients by using the *Quit* button on the *olwm* window menu. As well, some such as *xman*, have their own *Quit* button.

# 8.1 Desk Accessories

The clients *xclock*, *oclock*, *xcalc*, *xload*, *xbiff*, and *xman* can be thought of as *desk accessories*. "Desk accessories" is a term we've borrowed from the Macintosh environment, meaning small applications available—and useful—at any time. Many of the DeskSet clients described in Chapter 7, *The OpenWindows DeskSet Clients*, can also be considered to be Desk Accessories.

## 8.1.1 Clock Programs: xclock and oclock

The standard release of X includes two clients that display the time: *xclock* and *oclock* (the *oclock* client was added to the standard distribution of X in Release 4). The time displayed by both *xclock* and *oclock* is the system time set and displayed with the UNIX *date* (1) command, the MS-DOS *date* and *time* commands, etc., in other words, the operating system's notion of the correct time.

*xclock* continuously displays the time, either in analog or digital form, in a standard window. The analog *xclock* shows a round 12-hour clock face with tick marks representing the minutes. The digital *xclock* shows the 24-hour time (2:30 PM would be represented as 14:30) as well as the day, month, and year. You can run more than one clock at a time. The analog clock is the default. Figure 8-1 shows two *xclock* applications being run: an analog clock above a digital clock.

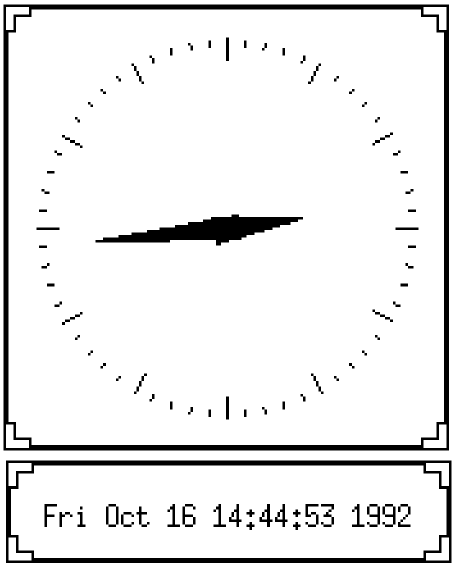



**Figure 8-1. Two xclock displays: analog clock above digital clock**

Note that *olwm* normally puts a titlebar on all windows. We suppressed this, to show how the *xclock* window would normally look, by adding them to the `MinimalDecor` list. For example, if you normally use *xclock*, you could add this to your *.Xdefaults* file:

```
OpenWindows.MinimalDecor: xclock
```

You don't have to do this, but it makes the *xclock* window look nicer if you do.

The *oclock* client (available as of X11 Release 4) displays the time in analog form on a round 12-hour clock face without tick marks. The only features of an *oclock* display are the round clock outline, hour and minute hands, and the "jewel" marking 12 o'clock.

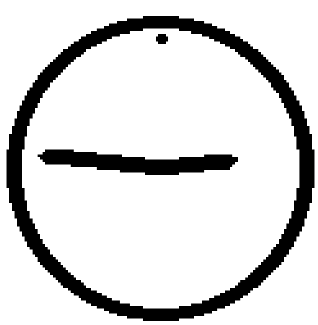


**Figure 8-2. oclock display**

Though it is somewhat more difficult to read the precise time on the simple *oclock* display, the *oclock* is perhaps a little more aesthetically pleasing than the analog *xclock*.

*oclock* also makes use of the X Shape server extension, which supports non-rectangular windows. If you try to resize the round *oclock*, you'll discover that it's possible to "stretch" it into various oblong shapes, as shown in Figure 8-3.

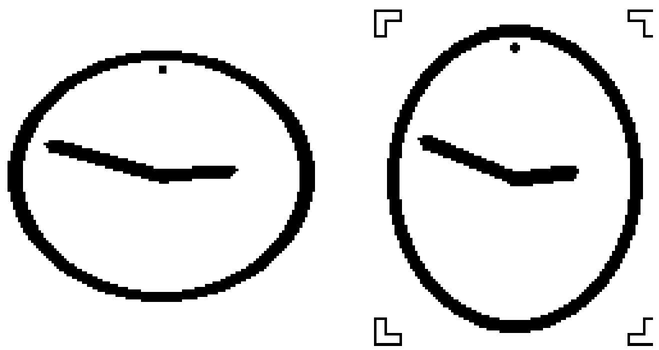


**Figure 8-3. Oblong oclock displays**

*oclock* sets some "window manager hints" to tell *olwm* that it doesn't need a titlebar, you don't need to add it to the `MinimalDecor` list. To move the *oclock* display under *olwm*, you need only grab it with the pointer. Click and hold SELECT, the left pointer button, and drag the clock to where you want it. To Resize it, however, is a two-step process:

- Click SELECT on the window's border, and the normal OPEN LOOK resize corners will appear, as in the right side of Figure 8-3.
- Then use these re-size corners as you would on any window.

This is how the two oblong clocks were produced.

Though the default colors for *oclock* are black and white, it was designed to be run in color. The minute hand, hour hand, jewel, clock border, and background can all be set to a color, using either command line options (described in Chapter 11, *Command-line Options*) or by specifying client resources (described in Chapter 12, *Setting Resources*). See the *oclock* reference page in Part Three of this guide for the necessary command line options and color suggestions.

## 8.1.2 A Scientific Calculator: xcalc

*xcalc* is a scientific calculator that can emulate a Texas Instruments TI-30 or a Hewlett Packard HP-10C.

OpenWindows users should also check out the operation of *calctool* described in Chapter 7, *The OpenWindows DeskSet Clients*.

Once you place the pointer within the *xcalc* window, the calculator can be operated in two ways: with the pointer, by clicking the first pointer button on the buttons in the calculator window, or with the keyboard, by typing the same numbers and symbols that are displayed in the calculator window. When using the first method, notice that the pointer appears as a small hand, enabling you to "press" the buttons.Figure 8-4 shows *xcalc* on the screen.

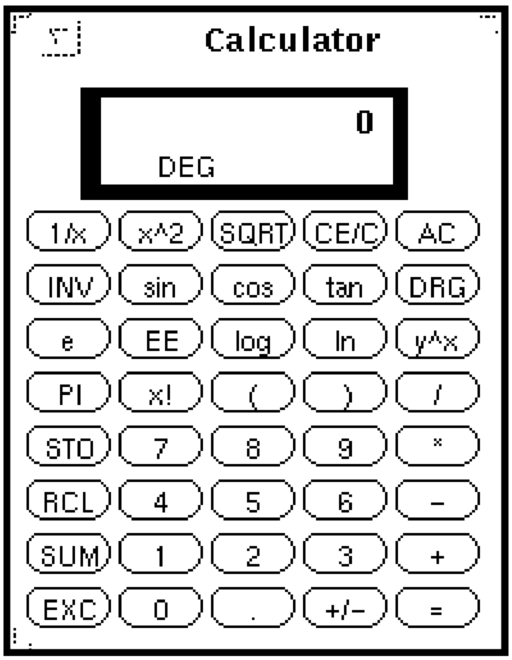



**Figure 8-4. The default xcalc (TI-30 mode) on the screen**

Figure 8-4 depicts the version of the calculator provided with Release 4 of X. As you can see, it features oval buttons. If you are running an earlier release, the calculator will have rectangular buttons and may also have a darker background color. We are discussing the Release 4 version. For additional information, see the *xcalc* reference page in Part Three of this guide.

The values punched on the calculator and the results of calculations are displayed in the long horizontal window along the top of the *xcalc*. You can enter values either by clicking on the calculator keys with the pointer, or by pressing equivalent keys on the keyboard. Most of the calculator keys have keyboard equivalents. The not-so-obvious equivalents are described on the *xcalc* reference page in Part Three of this guide.

By default, *xcalc* works like a TI-30 calculator. To run *xcalc* in this mode, enter:

```
% xcalc &
```

You can also operate the calculator in Reverse Polish Notation (as an HP-10C calculator operates), by entering:

```
% xcalc -rpn &
```

In Reverse Polish Notation the operands are entered first, then the operator. For example, `5 * 4 =` would be entered as `5 Enter 4 *`. This entry sequence is designed to minimize keystrokes for complex calculations.

*xcalc* allows you to select the number in the calculator display. You select the number using the first pointer button and paste it in another window using the second button. Since this is an Athena widget set application, you should refer to Appendix A, *The xterm/olterm Terminal Emulator*, for information about copying and pasting text selections.

*xcalc* can also be resized. In prior releases, this was not possible. Also as of X11R4, the code in *xcalc* to emulate an analog slide rule (`-analog`) is no longer available.

For more information on the function of each of the calculator keys, see the *xcalc* reference page in Part Three of this guide.

You can terminate the calculator by using the *Quit* menu button on the *olwm* window menu. For a method more in keeping with the spirit of this program, you can also terminate it by simulating "powering off" the calculator by clicking the right pointer button on the TI calculator's AC key or the HP calculator's ON key. You can even terminate it by positioning the pointer on the calculator and typing q, Q, or Control-C.

### 8.1.3 Mail Notification Client: xbiff

*xbiff* is a simple program that notifies you when you have mail. It puts up a window showing a picture of a mailbox. When you receive new mail, the keyboard emits a beep, the flag on the mailbox goes up, and the image changes to reverse video. Figure 8-5 shows the *xbiff* mailbox before and after mail is received. We didn't need to add *xbiff* to the `MinimalDecor` resource because its program sets certain "window manager hints" that tell *olwm* not to provide a titlebar.

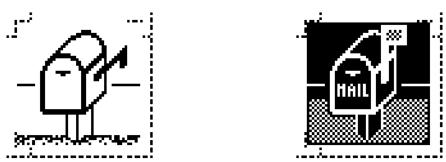



**Figure 8-5. xbiff before and after mail is received**

After you read your mail (and the mail file is empty), the image changes back to its original state; or you can click on the full mailbox icon with any pointer button to change it back to empty. Regardless of the number of mail messages when you do this, *xbiff* remembers the current size of your mail file to be the *empty* size.

This feature of *xbiff* has two implications, one obvious and the other not so obvious: when additional mail arrives (and the mail file becomes larger), *xbiff* notifies you; but when you delete messages from the current mail file (and the file becomes smaller), *xbiff* also notifies you—when you exit the mail program and the file is saved. This can be a little confusing until you get used to it.

While *xbiff* is intended to monitor a mail file, it can actually be set up to watch any file whose size changes using the `−file` option (followed by the name of the file to be monitored). For instance, if you're running a program that produces output intermittently, you can start *xbiff* with `−file` followed by the name of the output file; then *xbiff* will notify you when output is returned. (You can even specify an image other than the mailbox using resource variables—even for a single *xbiff* process.) See the *xbiff* reference page in Part Three of this guide for a list of options and resources. See Chapter 12, *Setting Resources*, for the syntax of resource specifications.

## 8.1.4 Monitoring System Load Average: xload

By default, *xload* polls the system for the load average at five-second intervals and displays the results in a simple histogram.

If you are running processes on more than one machine, it's useful to gauge the level of activity on the systems in question. This information should help you judge when to start processes and monitor how your processes are impacting system resources.

Say you're running clients both on the local machine *kansas* and on the remote machine *oz*. On the local display, you can have two *xload* windows, one showing activity on *kansas* and another showing activity on *oz*.

To create an *xload* window monitoring activity on *kansas*, use the command:

```
% xload &
```

Once the *xload* window is created, move it to a convenient location on the screen, using the pointer.

Then run an *xload* process on *oz* using a remote shell and display the results in a window on *kansas*:

```
% rsh −n oz xload −display kansas:0.0 &
```

The display option tells *xload* to create its window on the local display (*kansas*). Again, move the window using the pointer.

Figure 8-6 shows the resulting *kansas* display: two *xload* windows—the top window monitoring activity on the local system and the bottom one monitoring activity on the remote system.

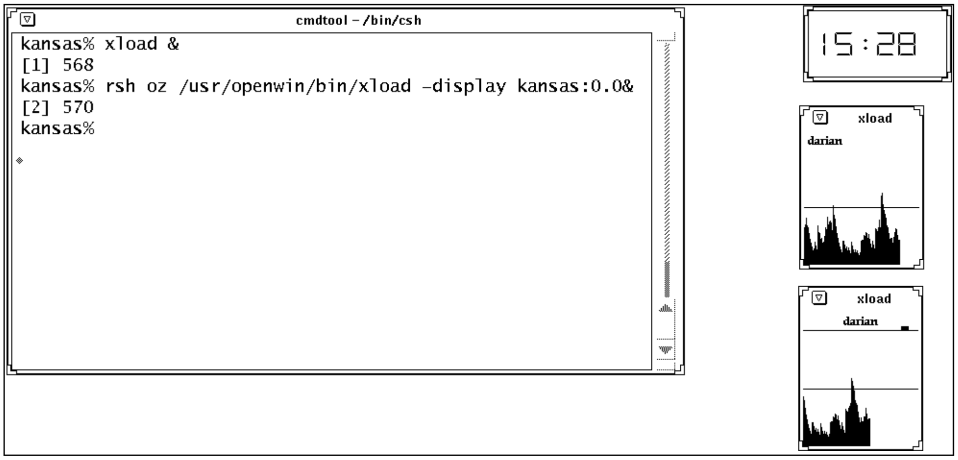



**Figure 8-6. Monitoring activity on two systems with xload**

For OpenWindows users, a simpler method is to use the more flexible *perfmeter* program, which knows how to reach out across the network to ask another machine for its load average (or any of almost a dozen statistics). *perfmeter* has the advantage that it gives a numerical scale to its graph, rather than the numberless graph that *xload* offers. There is more to *xload*, and much more to *perfmeter*, than these examples show. In particular, *perfmeter* lets you select CPU load, disk I/O, paging, ethernet traffic, and many other things to watch. And you can change the list of hosts you want to watch without quitting and restarting the program. See the description of *perfmeter* in Chapter 7, *The OpenWindows DeskSet Clients*. Also see the reference manual page in Section Three of this guide for more information about either *xload* or *perfmeter*.

## 8.1.5 Browsing Reference Pages: xman

The *xman* client allows you to display and browse through formatted versions of reference pages. By default, *xman* lets you look at the standard UNIX manual pages found in subdirectories of the directory */usr/man*. The standard version of X assumes there are 10 subdirectories: *man1* through *man8*, corresponding to the eight sections of reference pages in the UNIX documentation set; *manl* (man local) and *mann* (man new). You can specify other directories by setting the MANPATH system variable. (The individual directory names should be separated by colons.)

This section describes the version of *xman* provided with Release 4 of X. From a user's viewpoint, the general operation of the client has not changed much since prior releases but the organization of menus and options has changed.

Regardless of the version of X, you run *xman* by typing:

```
% xman &
```

in an *xterm* window.

The initial *xman* window, shown in Figure 8-7, is a small window containing only a few commands.


**Figure 8-7. Initial xman window**

This window is small enough to be displayed for prolonged periods during which you might have need to examine UNIX manual pages. You select a command by clicking on it with the first pointer button.

The *Manual Page* command brings up a larger window in which you can display a formatted version of any manual page in the MANPATH. By default, the first page displayed contains general help information about *xman*. Use this information to acquaint yourself with the client's features.* (The actual *xman* reference page in Part Three of this guide primarily describes how to customize the client.)

Once you've opened this larger window, you can display formatted manual pages in it.†

Notice the horizontal bar spanning the top edge of the window. (If you're running *olwm* or a similar window manager, this bar appears beneath the titlebar provided by the window manager.) The bar is divided into three parts labeled *Options*, *Sections*, and *Xman Help*. The part currently labeled *Xman Help* is merely informational and the text displayed in it will change depending on the contents of the window. The parts labeled *Options* and *Sections* are actually handles to two *xman* menus.

If you place the pointer on the *Options* box and press and hold down the first button, a menu called *Xman Options* will be displayed below. The menu is pictured in Figure 8-8.

The functionality of these options is described in the on-line *xman* help page. To select an option, move the pointer down the menu and release the first button on the option you want. The option you will probably use most frequently is the first one, *Display Directory*.

*Display Directory* lists the reference pages in the current reference page directory (also called a *section*). By default, this is *man1*, the user commands. When you list the contents

† Selecting the **Help** command also opens a large window in which the same help information is displayed. The **Help** command is something of a dead end, however; you cannot display any other text in this window.

Display Directory

Display Manual Page

Help

Search

Show Both Screens

Remove This Manpage

Open New Manpage

Show Version

Quit

**Figure 8-8. Xman Options menu**

of *man1* in this way, the informational section of the horizontal bar reads ***Directory of: (1) User Commands.***

*xman* displays each man page directory in a window known as a *viewport*, created with the Athena Viewport widget from the X Toolkit. A viewport is a composite widget that provides a main window and horizontal and/or vertical scrollbars.

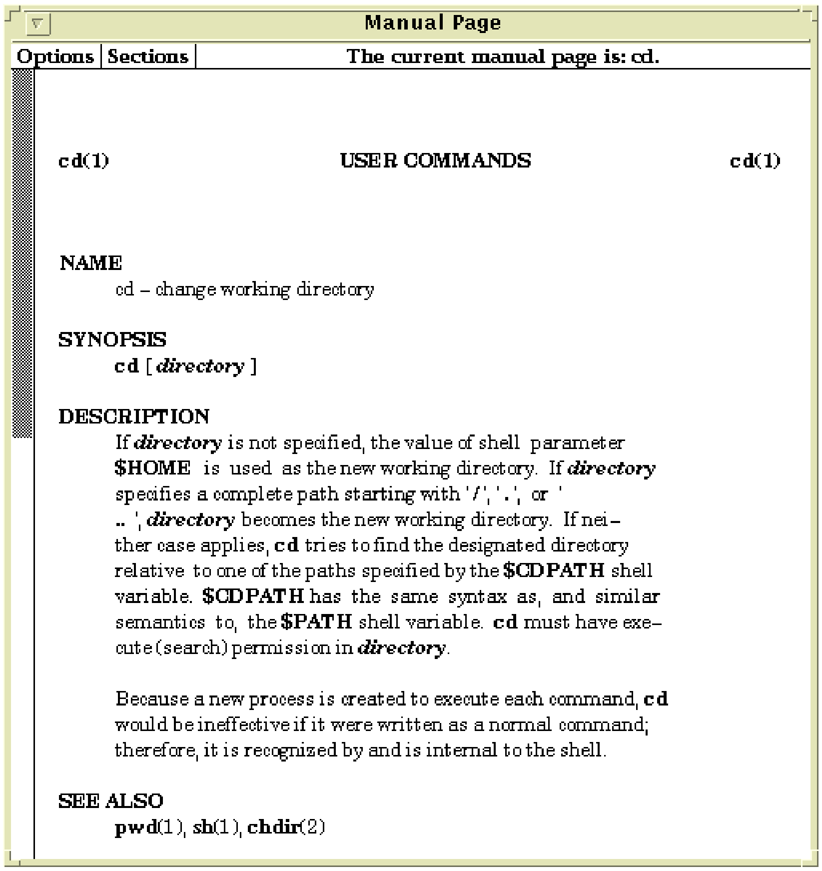
*xman*'s scrollbar is also an Athena widget. (See Appendix A, *The xterm/olterm Terminal Emulator*, for instructions on using an Athena scrollbar.) The OPEN LOOK Toolkit also provides a scrollbar, described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, which looks and operates differently than the Athena scrollbar.

For a list of the Athena widgets, see Appendix I, *Athena Widget Resources*. For complete information about the X Toolkit, see Volume Four, *X Toolkit Intrinsics Programming Manual*, and Volume Five, *X Toolkit Intrinsics Reference Manual*.

Once you've listed a reference page directory in the *xman* window, you can display a formatted version of any page in the list simply by clicking on the name with the first pointer button. Figure 8-9 shows the formatted reference page for the UNIX *cd* (1) command.

To display another manual page from the same directory, display the *Xman Options* menu again. Select *Display Directory* and the directory listing is again displayed in the window. Then click on another command name to display its manual page in the window. (If you decide not to display another reference page, you can remove the directory listing and go back to the reference page previously displayed by using the second *Xman Options* menu selection, *Display Manual Page*. *Display Directory* and *Display Manual Page* are toggles of one another.)

To display a manual page from another directory in the MANPATH, you must first change to that directory using the second *xman* menu, *Xman Sections*. Bring up the menu by placing the pointer in the *Sections* box in the application's titlebar and holding down the first button. The *Xman Sections* menu lists the default directories of UNIX manual pages (in this case for a Solaris 2 system), as shown in Figure 8-10.



Manual Page

Options | Sections | The current manual page is: cd.

cd(1) USER COMMANDS cd(1)

NAME
cd – change working directory

SYNOPSIS
**cd** [ ***directory*** ]

DESCRIPTION
If ***directory*** is not specified, the value of shell parameter **$HOME** is used as the new working directory. If ***directory*** specifies a complete path starting with ' / ', ' . ', or ' .. ', ***directory*** becomes the new working directory. If neither case applies, **cd** tries to find the designated directory relative to one of the paths specified by the **$CDPATH** shell variable. **$CDPATH** has the same syntax as, and similar semantics to, the **$PATH** shell variable. **cd** must have execute (search) permission in ***directory***.

Because a new process is created to execute each command, **cd** would be ineffective if it were written as a normal command; therefore, it is recognized by and is internal to the shell.

SEE ALSO
**pwd**(1), **sh**(1), **chdir**(2)

**Figure 8-9. cd reference page displayed in xman window**

Click on the first pointer button to select another directory of reference pages from which to choose. Once you select a directory, the files in that directory are listed in the window. Again, you display a page by clicking on its name with the first pointer button.

You can display more than one "browsing" window simultaneously by selecting the *Open New Manpage* option from the *Xman Options* menu. An additional reference page window will be opened again starting with the help information.

The various windows *xman* creates can all be iconified and each is represented by a different icon symbol. The icon symbols for the initial *xman* window, the help window, and the browsing window appear in Figure 8-11. Keep in mind that if you've displayed several browsing windows simultaneously, you can iconify each of them.

Note that the "Manual Page" icon label is too big to fit in the icon area provide by *olwm*. This is amusing but harmless.

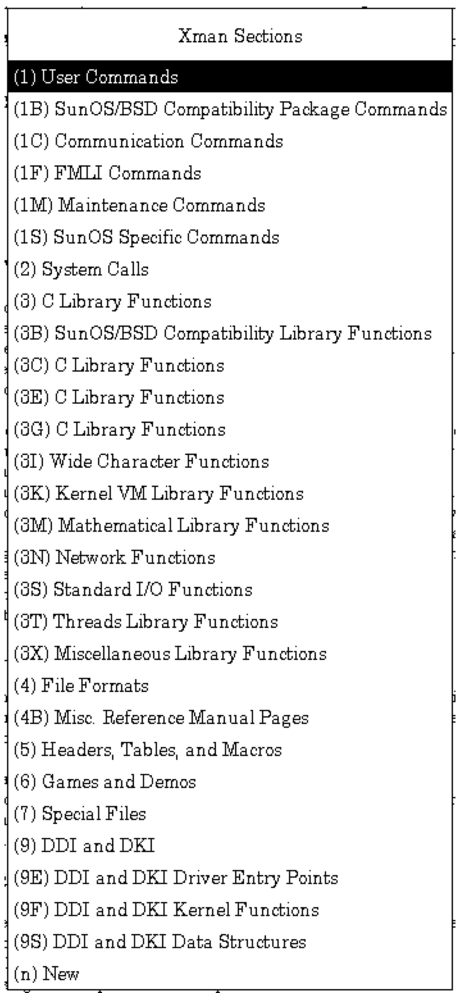



**Figure 8-10. Xman Sections menu**

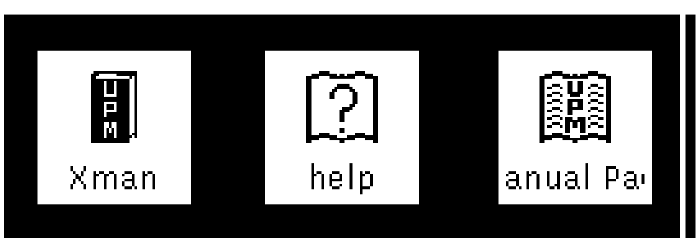



**Figure 8-11. Icons for xman's initial window, help window, and browsing window**

You can remove a browsing window by selecting the *Remove This Manpage* option from the *Xman Options* menu. Selecting *Quit* from the *Xman Options* menu or from the initial *xman* window causes the client to exit.

An alternate man page browser is *tkman*, which is included on the CD-ROM accompanying this book. The Tk toolkit provides a Motif-like interface, but *tkman* is a very useful browser - it can treat almost any text as a hyperlink, and jump to the appropriate man page.

OpenWindows users should also refer to *answerbook* and *helpviewer*, described in Chapter 7, *The OpenWindows DeskSet Clients*.

## 8.2 The xedit Text Editor

The *xedit* client provides a window in which you can create and edit text files. The editing commands *xedit* recognizes are provided by the Athena Text widget. Many other standard and user-contributed clients also include areas in which you can enter text. Several of these clients, including *xclipboard* and *xmh*, also use the Text widget, and thus recognize the same editing commands as *xedit*.

*xedit* is in some ways patterned after *textedit*, a version of which has been in SunOS for nearly a decade. OpenWindows users may wish to use *textedit*, described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, rather than *xedit*, since *textedit* uses the OPEN LOOK conventions..

The *xedit* client is valuable to illustrate the use of the Athena Text widget. (*xedit* can also be used to illustrate several other widgets.) However, we do not recommend using *xedit* as your primary text editor. The program's behavior can be erratic. For example, it's fairly easy to overwrite files inadvertently, as explained in the discussion of the *Load* button later in this section. The redraw command (Control-L) causes text in the window to scroll so as to reposition the cursor in the center of the editing window—not a welcome surprise. Some of the commands to create a new paragraph may also inadvertently copy preceding text. These are just a few of *xedit*'s inconvenient features. OpenWindows users, as mentioned, may wish to use *textedit*. A Motif-based free-software editor of similar power is the *asedit* program, source for which is included in your CD-ROM.

Still, it is necessary to know something about the Athena Text widget in order to be able to enter and edit text in windows provided by many standard X11 clients.

*xedit* recognizes various Control and Meta keystroke combinations that are bound to a set of commands similar to those provided by the *emacs* text editor.†

In addition, you can use the pointer to move the cursor in the text or to select a portion of text. The *xedit* cursor is a caret symbol (^). A caret cursor appears in each of the three areas that accept text entry. (These areas are described later in this section.) Pressing the first pointer button causes the insertion point (cursor) to move to the location of the pointer. Notice that the cursor always appears between characters, rather than on a character as the

† The commands may be bound to keys different from the defaults described below through the standard X Toolkit key translation mechanisms. See Chapter 12, *Setting Resources*, for more information.

*xterm* cursor does. Double-clicking the first pointer button selects a word, triple-clicking selects a paragraph, and quadruple-clicking selects everything. After you select text, the selection may be extended in either direction by using the third pointer button.

You can invoke *xedit* by entering:

```
% xedit &
```

Since no filename has been specified, the main section of the *xedit* window is empty, as illustrated by Figure 8-12.

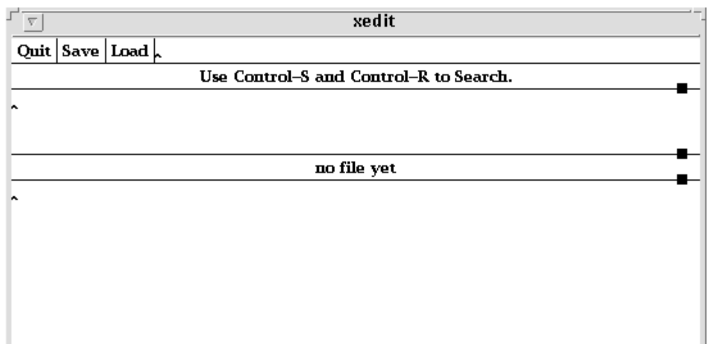



**Figure 8-12. xedit window before text file is read in**

Notice that the *xedit* window is divided into four parts:

- A commands section, which features three command push buttons (*Quit*, *Save*, and *Load*) and an area to their right in which a filename can be entered.
- A message window, which displays messages from the client and can also be used as a scratch pad.
- The filename display, which shows the name of the file being edited and the read/write permissions for the file.
- The edit window, in which the text of the file is displayed and in which you issue the editing commands.

The *xedit* application uses the Athena VPaned widget (of the X Toolkit), which arranges subwindows one above the other without overlapping. The subwindows are also known as *vertical panes* and the non-overlapping, top-to-bottom arrangement is commonly described as *vertical tiling*.

The individual panes organized by a VPaned widget can be any other type of widget. In the case of *xedit*, for example, the commands section is one pane that contains three command buttons (another widget) and a small window to the right of the buttons (a Text widget) in which a filename can be entered.

Notice the three small black rectangles on the borders between the panes. These features are called *grips* and they serve as handles to allow you to resize the subwindows. When the pointer is positioned on the grip and a button pressed, an arrow is displayed that indicates the direction in which the border between the two windows can be moved. If you move the pointer in the direction of the arrow (while pressing the button), one subwindow will grow while the other will shrink.

You can enter text in three areas of the *xedit* window: the message window, the edit window, and the small window immediately to the right of the command buttons in which you can enter a filename. (Thus all three use a Text widget.) Note that the small filename window to the right of the command buttons is different from the filename display (lower in the *xedit* window). The filename display is simply that—a display of the filename; the window does not support editing.

All three areas that permit editing display the caret text cursor. In order to focus keyboard input to a particular area, the pointer must rest in that area—regardless of whether *olwm* is operating with the default click-to-type focus. (If click-to-type focus is in effect, the *xedit* window must also be selected as the focus window.) This is extremely important to remember. Both the message window and the edit window will display a vertical scrollbar if the text is too large to fit. (Be aware also that a scrollbar technically is not part of the text entry window it borders. If the pointer is resting in a scrollbar, keyboard input will be lost—it will not be directed to the corresponding text area!)

The three push buttons in the commands section have the following functions:

*Quit* Exits the current editing session and closes the window. If changes have not been saved, *xedit* displays a warning in the message window, and does not exit, thus allowing the user to save the file.

*Save* Writes the file. If file backups are enabled (using the `enableBackups` resource), *xedit* first stores a copy of the unedited file as filename *.BAK* and then overwrites *filename* with the contents of the edit window. The *filename* used is the text that appears in the area immediately to the right of the *Load* button.

*Load* Loads the file displayed immediately to the right of the button into the edit window. If a file is currently being displayed and has been modified, a warning message will ask the user to save the changes, or to press *Load* again.

**This interface has at least two serious pitfalls**. First, if you're working on a file that has unsaved changes and you try to load a second file, it's possible to overwrite the second file. This is how it happens. In order to load a second file, you must enter the name of the file in the area next to the *Load* button; then press *Load*. If you try to load a second file while editing a file with unsaved changes, *xedit* warns you to save or press *Load* again. If you press *Save* the current file will be saved—but as the name to the right, the second file you intended to load.

If backups are not enabled, this action will overwrite the file you wanted to load. If backups are enabled, the first file will be saved under the name of the second file with a *.BAK* extension and the second file will not be overwritten. Because of this potential problem, we recommend that you set the resource `enableBackups` to `on` (and load the resources using *xrdb*) before using *xedit*.

A second problem can occur after you've loaded a file by entering the name in the window next to the *Load* button. Say you've been editing the file for some time, but haven't saved the changes. If you go to save the changes and accidentally double-click on *Load* (not that difficult to do), you'll reload the version of the file before you made the edits. The changes are lost!

Now, after considering some of the possible pitfalls, let's load a file into the empty *xedit* window as shown in the figure above. (Obviously, in this case, there's no danger of over-writing an existing file.) To load a file called *index.html*:

1. Place the pointer in the area to the right of the `Load` button.

2. Type *index.html*. The caret cursor moves as you type.

3. Place the pointer on the *Load* command button and press the first pointer button.

The file called *index.html* is displayed in the edit window, as shown in Figure 8-13.

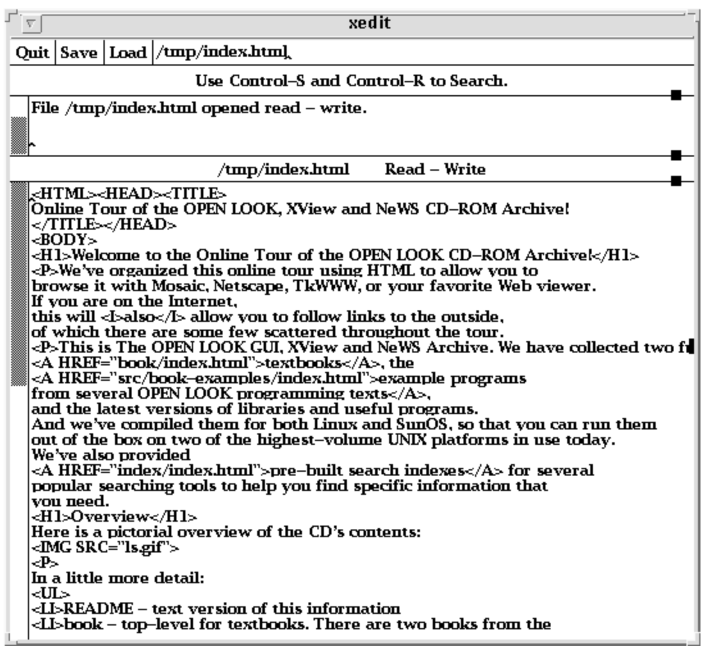



**Figure 8-13. text file displayed in xedit window**

The simpler commands to edit or append text are intuitive. A backspace deletes the character to the left of the cursor. Typing enters characters immediately before the cursor point, causing the cursor to advance to the right. When you first load a file, the cursor appears at the beginning of the text in the edit window. If you want to append text to the end of the file, move the pointer to the end of the text and click the first button. The caret cursor appears where the pointer is and any text you type is added to the end of the file, moving the cursor to the right.

The list at the end of this section summarizes all of the editing command recognized by *xedit*. In this list of commands, a *line* refers to one row of characters displayed in the window. A *paragraph* refers to the text between manually inserted carriage returns or blank lines. Text within a paragraph is automatically broken into lines based on the current width of the window.

The keystroke combinations are defined as indicated. (Note that "Control" and "Meta" are two of the "soft" keynames X recognizes. They are mapped to particular physical keys which may vary from keyboard to keyboard. See the "xmodmap" section in Chapter 14, *Customization Clients*, for a discussion of modifier key mapping.) If you are using an earlier release of X, a few of these keystroke combinations may produce slightly different results.

Keep in mind that you can redefine any of these key combinations using what are known as *translations*. Translations allow you to assign actions recognized by a client to particular key combinations, or key and pointer button combinations. For example, *xedit* recognizes actions to delete text, to copy text, to move the cursor, etc. *xedit* defines key combinations to invoke these actions. (The key/action mappings appear in the list at the end of this section.) For information on specifying alternate mappings, see "Event Translations" in Chapter 12, *Setting Resources*.

Note that the function assigned to the Return key in the following list applies only to the edit window and message window. In the filename window (next to the command buttons), Return simply moves the cursor to the end of the line.

**Table 8-1. Xedit key bindings**

| Key sequence | Function |
|---|---|
| Control-A | Move to the beginning of the current line. |
| Control-B | Move backward one character. |
| Control-D | Delete the next character. |
| Control-E | Move to the end of the current line. |
| Control-F | Move forward one character. |
| Control-H or Backspace | Delete the previous character. |
| Return, Control-J, Control-M, or LineFeed | New paragraph. (**Linefeed**, **Control-J**, and **Control-M** may be unreliable on some terminals.) |

Table 8-1. Xedit key bindings

| Key sequence | Function |
|---|---|
| Control-K | Kill the rest of this line. (Does not kill the carriage return at the end of the line. To do so, use **Control-K** twice. However, be aware that the second kill overwrites the text line in the kill buffer.) |
| Control-L | Redraw the window. (Also scrolls text so that cursor is positioned in the middle of the window.) |
| Control-N | Move down to the next line. |
| Control-O | Divide this line into two lines at this point. |
| Control-P | Move up to the previous line. |
| Control-V | Move down to the next screenful of text. |
| Control-W | Kill the selected text. |
| Control-Y | Insert the last killed text. (If the last killed text is a carriage return—see Control-K above—a blank line is inserted.) |
| Control-Z | Scroll up the text one line. |
| Meta-< | Move to the beginning of the file. |
| Meta-> | Move to the end of the file. |
| Meta-[ | Move backward one paragraph. |
| Meta-] | Move forward one paragraph. |
| Meta-B | Move backward one word. |
| Meta-D | Kill the next word. |
| Meta-F | Move forward one word. |
| Meta-H or Meta-Delete | Kill the previous word. |
| Meta-I | Insert a file. If any text is selected, use the selected text as the filename. Otherwise, a dialog box will appear in which you can type the desired filename. |
| Meta-V | Move up to the previous screenful of text. |

**Table 8-1. Xedit key bindings**

| Key sequence | Function |
|---|---|
| Meta-Y | Insert the last selected text here. Note that this can be text selected in some other text subwindow. Also, if you select some text in an *xterm* window, it may be inserted in an *xmh* window with this command. Pressing pointer button 2 is equivalent to this command. |
| Meta-Z | Scroll down the text one line. |
| Delete | Delete the previous character. |

# 8.3 Window and Display Information Clients

The standard release of X includes four clients that provide information about windows on the display and about the display itself. Much of the information is probably more relevant to a programmer than to the typical user. However, these clients also provide certain pieces of information, such as window geometry, window ID numbers, and the number and nature of screens on the display, that can assist you in using other clients.

## 8.3.1 Displaying Information about a Window: xwininfo

The *xwininfo* client displays information about a particular window. Some of this information can be useful in determining or setting window geometry (described in Chapter 3). *xwininfo* also provides you with the *window ID* (also called the resource ID). Each window has a unique identification number associated with it. This number can be used as a command line argument with several clients. Most notably, the window ID can be supplied to the *xkill* client to specify the window be killed.

You can also use the window ID as an argument to the *xprop* client, which displays various window properties. As described in Chapter 1, a property is a piece of information associated with a window or a font and stored in the server. Properties facilitate interclient communication; they are used by clients to store information that other clients might need to know. Storing properties in the server makes the information they contain accessible to all clients. See Chapter 1, the *xprop* reference page in Part Three of this guide, and Volume One, *Xlib Programming Manual*, for more information about properties and the *xprop* client.

To display information about a window, type this command in an *xterm* window:

```
% xwininfo
```

The pointer changes to the cross-hair pointer and you are directed to select the window about which you want information:

```
xwininfo ==> Please select the window about which you
==> would like information by clicking the
```

```
==> mouse in that window.
```

You can select any window on the display, including the window in which you've typed the command and the root window. (Rather than using the pointer, you can specify a window on the command line by supplying its title, or name if it has no title, as an argument to *xwininfo*'s own `-name` option. See Chapter 9 for information about setting a client's title and name. See the *xwininfo* reference page in Part Three of this guide for a list of its options.)

The following diagram, Figure 8-14, shows the statistics of *xwininfo* supplies with some typical readings.

```
xwininfo ==> Window id: 0x70000e (xterm)
==> Absolute upper-left X: 12
==> Absolute upper-left Y: 29
==> Relative upper-left X: 0
==> Relative upper-left Y: 0
==> Width: 818 ==> Height: 484
==> Depth: 8 ==> Border width: 0
==> Window class: InputOutput
==> Colormap: 0x8006b (installed)
==> Window Bit Gravity State: NorthWestGravity
==> Window Window Gravity State: NorthWestGravity
==> Window Backing Store State: NotUseful
==> Window Save Under State: no
==> Window Map State: IsViewable
==> Window Override Redirect State: no
==> Corners: +12+29 -322+29 -322-387 +12-387
```

**Figure 8-14. Window information displayed by xwininfo**

These readings are for a login *xterm* window displayed using a 12-point Roman Courier font. All numerical information is in pixels, except depth, which is in bits per pixel. The *olwm* window manager is also running. The most significant statistics from the above figure for the average user are:

```
xwininfo ==> Window id: 0x70000e (xterm)
==> Absolute upper-left X: 12
==> Absolute upper-left Y: 29
==> Relative upper-left X: 0
==> Relative upper-left Y: 0
==> Width: 818 ==> Height: 484
==> Depth: 8 ==> Border width: 0
==> Colormap: 0x8006b (installed)
==> Corners: +12+29 -322+29 -322-387 +12-387
```

The first piece of information is the window ID, which can be used as an argument to *xkill*. Specifying the window to be killed by its ID number is somewhat less risky than choosing it with the pointer.

With many window managers, you can use some of the other statistics to gauge the window's geometry (size and position). Generally, the absolute upper-left X and Y correspond to the positive x and y offsets that can be supplied to the `-geometry` option used to place a client window. (The use of the `-geometry` option is discussed in Chapter 3, *Opening Additional Windows*.).

The Window Manager window frame complicates matters. When *olwm* or another framing window manager such as *mwm* is running, the absolute upper-left X and Y correspond to the x and y coordinates of the application window—but not the framed window!

Let's take another look at the sample *xwininfo* output. The absolute upper-left X and Y suggest that the window is located at coordinates 12,29. However, the output is actually for an *xterm* located at coordinates 0,0! The 12,29 are the coordinates of the *xterm* window itself; the coordinates represent the distance of the window from 0,0 *including the dimensions of the frame*. The default frame for *mwm* is actually 12 pixels in the x dimension and 29 pixels in the y dimension (because of the titlebar).

For any given window manager and set of font and size resources, there will be some fixed offset (like the 12,29 above) that you can subtract to get the actual position.These figures can be supplied as arguments to the `-geometry` option on the command line to specify window placement, as described in Chapter 3, *Opening Additional Windows*. (Chapter 3 also describes two simpler method of gauging x and y offsets.)

The relative upper-left X and Y may or may not be meaningful depending upon which window manager you are using. With *twm*, for example, regardless of a window's location, the relative upper-left X and Y are 0 and 0.

The four corners (again, including the frame) are listed with the upper-left corner first and the other three clockwise around the window (i.e., upper-right, lower-right, lower-left). The coordinates of the upper-left corner are, of course, the absolute upper-left X and Y. The width and height in pixels are somewhat less useful, since the geometry option to *xterm* requires that these figures be specified in characters and lines.

The values for window depth and colormap relate to how color is specified. See the discussion of color in Chapter 12, *Setting Resources*, for more information.

The other statistics provided by *xwininfo* are listed below:

```
==> Window class: InputOutput
==> Window Bit Gravity State: NorthWestGravity
==> Window Window Gravity State: NorthWestGravity
==> Window Backing Store State: NotUseful
==> Window Save Under State: no
==> Window Map State: IsViewable
==> Window Override Redirect State: no
```

These statistics have to do with the underlying mechanics of how a window is resized, moved, obscured, unobscured, and otherwise manipulated. They are inherent in the client program and you cannot specify alternatives. For more information on these and other window attributes, see Chapter 4 in Volume One, *Xlib Programming Manual*.

You can also use *xwininfo* with various options to display other window attributes. See the reference page in Part Three of this guide for details.

## 8.3.2 Listing the Window Tree: xlswins

Windows are arranged in a hierarchy, much like a family tree, with the root window at the top. The *xlswins* client displays the window tree starting with the root window, listing each window by its resource ID and title (or name), if it has one. (See Chapter 9 for a discussion of setting a client's title and name with command line options.)

A resource ID can be supplied to *xkill* to specify the window to kill. You can also supply a resource ID to *xwininfo* to specify the window you want information about, or to *xprop* to get the window's properties. Being able to display the ID numbers of all windows on the screen simultaneously is especially helpful if one or more windows is obscured in the stack. The *xwininfo* client is virtually useless in situations in which one window is hidden behind another. *xlswins* allows you to determine by process of elimination which window is hidden—without having to circulate all the windows on your screen. You can then use *xwininfo* with the ID number (displayed by *xlswins*) to get information about the obscured window.

Figure 8-17 shows the results of *xlswins* for a simple window arrangement: a single *xterm* (login) window on a root window. (No window manager is running.)

```
0x8006e ()
 0x30000e (xterm)
 0x300015 ()
 0x300016 ()
```

**Figure 8-15. Window tree displayed by xlswins**

The *xterm* window is easily identified. Any client that displays an application window, such as *xterm*, *xclock*, *xfd*, *bitmap*, etc., will be listed by name (in parentheses) following the ID number†. The root window is listed above the *xterm* in the window hierarchy. Client (and other) windows displayed on the root window are called *children* of the root window, in keeping with the family tree analogy; thus, the root window is the parent of the *xterm* window. In the *xlswins* listing, a child window is indented once under its parent.

But what are the other windows listed in Figure 8-15? A superficial examination of these other windows provides a brief introduction to the inner workings of X. An underlying feature of X is that menus, boxes, icons, and even *features* of client windows, such as scrollbars, are actually windows in their own right. What's more, these windows (and client window icons) may still exist, even when they are not displayed.

The two remaining windows are unnamed. From the relative indents of the windows, we can tell certain information. The first unnamed window is a child of the *xterm*, the second is a child of the child.

If we again run *xlswins*, this time requesting a long listing (with the `-l` option), we get geometry information that helps identify each window. This is shown in Figure 8-16.

† Most likely, you will not have to deal with the ID numbers for windows other than the explicitly named client windows. You can use the IDs of the client windows in all of the ways we've discussed: with *xkill*, *xwininfo*, *xprop*, etc.

```
0: 0x8006e (); ()() 1152x900+0+0 +0+0
1: 0x30000e (xterm); (xterm)(XTerm) 818x484+0+0 +0+0
2: 0x300015 (); ()() 818x484+0+0 +1+1
3: 0x300016 (); ()() 14x484+-1+-1 +0+0
```

**Figure 8-16. Window tree with geometry specifications**

The first number on each line refers to the level of the window in the hierarchy, the root window being at level 0, client windows at 1, etc. Following the *xterm* application window are what are known as the instance and class resource names for the client (in this case, `xterm`, `XTerm`). You use the instance and class resource names to specify default window characteristics, generally by placing them in a file in your home directory. This is described in detail in Chapter 12, *Setting Resources*.

The first geometry string is the complete specification relative to the *parent* window. The second geometry string is the current position relative to the *root* window. Since *olwm* is not running, frames are not an issue. Thus, a window at coordinates 0,0 would have the position `+0+0` relative to the root.

The two unnamed windows under *xterm* are the VT102 window and the window's scrollbar, respectively. (The first *xterm* listing is the application shell window, which can be displayed both as a VT102 and a Tektronix window.)

The listing in Figure 8-16 was generated when no window manager was running. If *olwm* is running, the *xlswins* output is considerably more complicated. Many of the features provided by *olwm*, such as the window frame and its command buttons, and the *Root Menu* and *Window Menu*, are actually windows themselves. This *greatly* complicates the window hierarchy. If you run *xlswins* while *olwm* is running, even if the display has only a single application window, the output will be dozens of lines long; you can assume that most of the mysterious windows in the hierarchy are features provided by the window manager. We just ran *xlswins* on a system with one *cmdtool*, one *contool*, and a publishing software package with one document open, another iconified, and three popups pinned up. There were so many windows that we had to use the standard UNIX utility *wc* (word count) to count them all:

```
% xlswins | wc -l
 149
%
```

You may also notice that application windows, such as *xterm*, are now at level 3 in the hierarchy. This is because *olwm reparents* all client windows; that is, the window manager creates another window that is the parent window of the application window and is itself the child of the root window. (The frame is actually a window in its own right; think of the window manager as creating a window that contains the application window.)

The geometry strings for application windows will also be different when *olwm* is running because of this reparenting and because of the presence of the frame. The first geometry string, which gives the position relative to the parent window, will always end with the *x,y* coordinates +0+0, since the parent is the window manager. The second geometry string, which gives the position relative to the root window, will include the dimensions of the frame. A window located at coordinates 0,0 will have the string +12,+29 because the x and

8

y dimensions of the default frame are 12 and 29 pixels, respectively. See the preceding discussion of *xwininfo* for more information.

For more information on the window hierarchy, see Volume One, *Xlib Programming Manual*.

### 8.3.3 Listing the Currently Running Clients: xlsclients

You can get a listing of the client applications running on a particular display by using *xlsclients*. Without any options, *xlsclients* displays a two-column list, similar to:

```
colorful xterm -geometry 80x24+10+10 -ls
colorful xclock -geometry -0-0
```

The first column shows the name of the display (machine) and the second the client running on it. The client is represented by the command line used to initiate the process.

This sample listing indicates that there is one *xterm* window and one *xclock* window running on the display `colorful`. (The option `-ls` following the *xterm* command reveals that the shell running in this window is a login shell.)

You can use *xlsclients* to create an *.xsession* or *.xinitrc* file, which specifies the clients you want to be run automatically when you log in. In order to do this, you must have set up client windows in an arrangement you like using command line options alone (that is, without having moved or resized windows via the window manager). You can then run *xlsclients* to print a summary of the command lines you used to set up the display and include those command lines in your *.xsession* or *.xinitrc* file. See "Customizing your Session Start-up" on Page 72, for information on configuring your session.

By default, *xlsclients* lists the clients running on the display corresponding to the DISPLAY environment variable, almost always the local display. You can list the clients running on another display by using the `-display` command line option. See Chapter 3, *Opening Additional Windows*, for more information about the `-display` option.

With the option `-l` (indicating long), *xlsclients* generates a more detailed listing. Figure 8-17 shows the long version of the listing on the previous page.

```
   Window 0x30000e:
 Machine: colorful
 Name: xterm
 Icon Name: xterm
 Command: xterm -geometry 80x24+10+10 -ls
 Instance/Class: xterm/XTerm
Window 0x40000b:
 Machine: colorful
 Name: xclock
 Icon Name: xclock
 Command: xclock -geometry -0-0
 Instance/Class: xclock/XClock
```

**Figure 8-17. Long xlsclients listing**

For each client, *xlsclients* displays six items of information: the window ID number, machine name, client name, icon name, command line used to run the client, and the instance and class resource names associated with the client.

As we'll see in Chapter 9, many clients, including *xterm*, allow you to specify an alternate name for a client and a title for the client's window. If you've specified a title, it will appear in the *xlsclients* `Name` field. If you haven't specified a title but have specified a name for the application, the name will appear in this field. Neither of the clients in the sample display has been given an alternate name or title.

You use the instance and class resource names to specify default window characteristics, generally by placing them in a file in your home directory. This is described in detail in Chapter 10, *Setting Resources*.

### 8.3.4 Listing the Currently Running OpenWindows Clients: psps

*psps* is an OpenWindows-specific tool to list the currently-running xnews clients, both those talking the X11 protocol and those using the NeWS protocol. Its output will be useful primarily to those developing NeWS applications, since there is, for example, no documented way to kill a NeWS client by its process id. In particular, the "ID" numbers do *not* correspond to the X11 "resource ID" used with *xkill* described above; trying to use the PS ID as the `-id` argument to *xkill* results in a "BadValue' X Error, indicating that X11 does not recognize the ID as a resource ID.

Here is an example output:

```
% psps
 ID State Pri ESS OSS DSS Name
> 0x2e0088 runnable 0 6 3 2 psps
> 0x2e2218 IO_wait 100 2 0 3 darian X11 client
> 0x2de218 IO_wait 100 2 0 3 darian X11 client
> 0x232218 runnable 100 2 0 3 darian X11 client
> 0x2c2dbc input_wait 0 2 0 2 RoundBaseFrame
> 0x236218 IO_wait 100 2 0 3 darian X11 client
> 0x212218 IO_wait 100 2 0 3 darian X11 client
> 0x19e218 IO_wait 100 2 0 3 darian X11 client
> 0x204218 IO_wait 100 2 0 3 darian X11 client
> 0x206218 IO_wait 100 2 0 3 darian X11 client
> 0x1f4218 IO_wait 100 2 0 3 darian X11 client
> 0x1f6218 IO_wait 100 2 0 3 darian X11 client
> 0x1ed440 IO_wait 100 4 3 1 X Listener %socketl/tmp/.X11-unix/X0
> 0x1edb5c IO_wait 100 4 3 1 X Listener %socketl6000
> 0x1e21e0 IO_wait 100 4 3 1 NeWS Listener unix
> 0x1e0c54 IO_wait 100 4 3 1 NeWS Listener tcp
> 0x1d638c input_wait 100 6 1 2 Global System EventMgr
> 0x1d79f8 input_wait 0 6 1 2 Global UI EventMgr
> 0x1be500 zombie 0 0 0 0 nullprocess
> 0x199e88 zombie 0 0 0 0 NullCanvasEventMgr
%
```

As you can see, the output resembles a UNIX *ps* command output, hence the name *psps* (PostScript Process Status). It is in fact a listing of the "lightweight processes" running inside the X/NeWS server program.

The last column is a textual description of the program; X11 clients are listed by the host they are from, but are not further identified. NeWS clients are identified by the type of their top-level window; *RoundBaseFrame* is the round NeWS clock shown in several of the examples in this book. Details of the other columns may be found in the reference page in Section Three of this guide.

### 8.3.5 Getting Information about the Display: xdpyinfo

The *xdpyinfo* client gives information about the X display, including the name of the display (contents of the DISPLAY variable), version and release of X, number of screens, current screen, and statistics relating to the color, resolution, input, and storage capabilities of each screen. The *xdpyinfo* reference page in Part Three of this guide shows a listing for a display that supports both a color and monochrome screen.

Much of the information provided by *xdpyinfo* has to do with how clients communicate information to one another and is more relevant to a programmer than to the typical user. However, the basic statistics about the name of the display, the version and release of X, and the number and nature of screens might be very helpful to a user, particularly one who is using a display for the first time.

The output of *xdpyinfo* appears on its "standard output", normally it appears in the terminal emulator window from which you started it. The output is voluminous; you probably want to pipe it through *more* or *pg*.

In addition, the detailed information about each screen's color capabilities can also be very valuable in learning how to use color more effectively. This information includes the default number of colormap cells: the number of colors you can use on the display at any one time. See Chapter 12, *Setting Resources*, for more information on the use of color and how to specify colors for many clients.

If you are of a programmatic bent, refer to Volume One, *Xlib Programming Manual*, for insights into some of the other information provided by *xdpyinfo*.

## 8.4 Killing a Client Window

You can normally kill any client with the *Quit* item at the bottom of the OPEN LOOK Window Manager's *Window* menu. For this reason, most OPEN LOOK clients do not provide a special *Quit* button or command. Programs such as terminal emulators, of course, can be terminated by exiting the shell in them, either by the *exit* keyword or by typing your EOF character (normally CTRL/D). However, there are a few cases when you need something more drastic; these should only be used as a last resort, as they *are* drastic. *xkill* kills an X client, and *pam* hides (but doesn't kill off) a NeWS window. These are described in the following sections.

Generally, you should exhaust the safer alternatives before you use *xkill* and other commands that kill a client. When you want to remove a window, depending on the client and what commands it recognizes, try these methods (roughly) in this order:

1. The *Quit* item on the window menu
2. Methods that cause the client to exit after finishing relevant processes:

- Special commands (e.g., *exit*) or key sequences (e.g., Control-D, Control-C, q, Q) recommended to stop a client.
- Certain application-specific menu items (e.g., for *xterm*, the **Main Options** menu commands **Send HUP Signal**, **Send TERM Signal**, and **Quit**).
- The *Quit* button on those applications which provide one

3. When these methods don't work, *then* use commands or menu items that kill the client:

- The **Send KILL Signal** item on the *xterm* **Main Options** menu, for removing *xterm* windows only. (See Appendix A, *The xterm/olterm Terminal Emulator*.)
- The UNIX *kill* command with the client's process ID number, which is determined using *ps*. (This method of removing a window is described for *xclock* earlier in this chapter.*)†
- The *xkill* client (described below) or, for NeWS clients, *pam* (see Section 8.4.2, "Killing a window with pam NeWS-based (OpenWindows only).

### 8.4.1 Killing a Client with xkill

The *xkill* program allows you to kill a client window or, more specifically, to force the server to end the connection to the client. The process exits and the associated window is removed.

*xkill* is a fairly drastic method of terminating a client and should *not* be used as the method of choice. In most cases, clients can be terminated in other ways. The possible repercussions of using *xkill* and some of the alternatives are discussed in the next section.

*xkill* is intended primarily to be used in cases where more conventional methods of removing a client window do not work. It is especially useful when programs have displayed undesired windows on the screen. To remove a stubborn client window, type:

```
% xkill
```

on the command line of an *xterm* window. The pointer changes to a "draped box" pointer and you are instructed to:

```
Select the window whose client you wish to kill with button 1 . . .
```

† This method is powerful but in practice has limitations. UNIX only allows you to kill a process if you are the owner of the process or if you are root. Thus, if a client has been started on your display from a remote system and you don't know the root password, you may not be in a position to use the UNIX *kill* command.

Move the draped box pointer to the window you want to remove, as shown in Figure 8-18, and click the first pointer button. The window is removed. (*xkill* does not allow you to select the root window.)

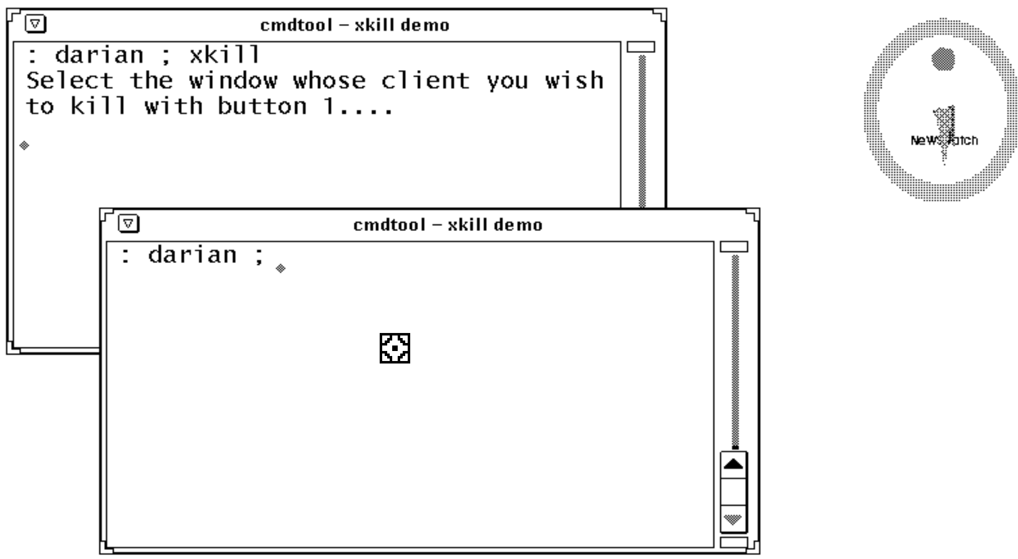



**Figure 8-18. Selecting the window to be removed.**

You can also specify the window to be killed by its *window ID* (also called the *resource ID*). Every window has an identification number associated with it. The *xwininfo* client can be used to display a window's window/resource ID (see the section "Window and Display Information" later in this chapter).

To remove a window using its ID number, type:

```
% xkill -id number
```

The window with the ID ***number*** is removed. Killing a window by its ID number is more cumbersome but it's somewhat safer than choosing the window to be killed with the pointer. It's too easy to click in the wrong place. (Of course, it's less treacherous to use the pointer on an isolated window than a window in a stack.)

Do not use *xkill* on an icon window, as they are always created by the window manager, not the application; if you do use xkill on the icon, therefore, you will kill your window manager and not the application!

## 8.4.2 *Killing a window with pam* NeWS-based (OpenWindows only)

It sometimes happens that a NeWS client will die and leave its window up on the screen. To "hide" windows permanently, there is an OpenWindows tool called "pam".

It works like *xkill* in that it gives you a special cursor that you click on a window, and gives you the message (on standard output):

```
Click on the stuck window to spray some Pam (tm) on it...
```

When it has hidden the window, it says:

```
Unstuck! Remember, you can't see it, but it's still there.
```

to remind you that any resources the window was consuming are still in use. Further, if you use *pam* on an X window that is active, the window is simply unmapped; the process is *not* killed; you would then have to use the UNIX *kill* command to terminate it, as there is no good way to re-map a *pam*med X11 window.

## 8.5 Demonstration Programs and Games

In one sense, many of the standard MIT clients are "demonstration" programs as they are intended to demonstrate the Athena widgets, both to users and to programmers. But some are specifically written to demonstrate aspects of The X Window System, and others are games for the pure and simple fun of playing.

**Table 8-2. X11 Demonstrations and Games**

| Name | Function | Notes |
|---|---|---|
| plaid | | |
| puzzle | | |
| | | |
| | | |
| MORE | TO | COME |
| | | |

We've now looked at most of the standard clients in a distribution of The X Window System. Now we turn our attention to some special-purpose graphics clients, both in standard X and in the OPEN LOOK distributions.