CHAPTER 1

An Introduction to OPEN LOOK and X Windows

This chapter describes the features of a typical display running OPEN LOOK on the X Window System, and introduces some basic window system concepts. It also provides an overview of the X Window System's client-server architecture and briefly describes some common clients.

The OPEN LOOK Graphical User Interface (GUI) gives display-based programs an interface that is efficient, easy to use, and consistent. OPEN LOOK is supported by Sun, by AT&T, by many system and application vendors, and is the standard graphical interface with the current release of UNIX, System V Release 4.

The X Window System, called X for short, is a network-based graphics window system that was first developed at MIT around 1984. Several versions of X have been developed, the most recent of which is X Version 11 (X11), first released in 1987.There have been several revisions of X11 since then. The latest is Release 6, or "X11R6".

X11 has been adopted as an industry standard windowing system. It's supported by every significant workstation vendor, and its development is now managed by The X Consortium, a group of UNIX and X industry leaders such as Sun, DEC, Hewlett-Packard, IBM, NCD, and many others who have united to direct, contribute to, and fund its continuing development. In addition to the system software development directed by the X Consortium, many programmers develop "contributed" or free software for X, and many software developers are producing commercial application software specifically for use with X, including spreadsheets, graphics and database programs, and publishing applications.

But the original versions of X11 did not provide or specify any graphical user interface. X is a basic window system upon which almost any style of GUI can be built. Even "The X Toolkit" provides only a simplified approach to creating graphical user interface components—guidelines for writing and implementing widgets—rather than offering a set of components with a predefined look and feel.

When X first started to become popular, in fact, it was so ugly that one old-timer said: "X [as it was then, with] each program having its own randomly-chosen conventions, looks like a ransom-note user interface system. This was certain to drive people away from X and UNIX to the Macintosh."

The X developers at MIT took a hands-off approach; they did not want to step into the GUI standards vacuum. Their battle cry was (then, and for many years after) "Mechanism, not Policy." That is, X itself has, and will always have, the flexibility to support any number of user interface styles.

To fill the standards gap, Sun and AT&T developed the OPEN LOOK GUI specification with input from many others in the industry. The OPEN LOOK document specifies not only the shape of various buttons and gadgets on the screen, but also the operation of various important clients, such as the File Manager (see Chapter 2, *Working in the OPEN LOOK Environment*, and Chapter 4, *Using the OPEN LOOK File Manager*). They published the specification quite early, so that other vendors could work to it, making OPEN LOOK the leading open standard for graphical user interface.

The design they came up with is clean, simple, and intuitive. My son Ben, then three and a half years old, climbed up on my lap one day, pointed at the scrollbars (graphic objects used to move or "scroll" a large data window around in a smaller on-screen window). Without puzzling over graphic design for a second, he asked me "What are those elevator things for"? And before long he was rearranging the windows on my screen. So even a three year old can learn to use OPEN LOOK: the design is clean, simple, and intuitive.

Shortly after Sun and AT&T announced that they would be working together on the development of OPEN LOOK, the Open Software Foundation announced its own development of a "competing standard" GUI. This led to the eventual release of OSF/Motif, comprising a set of "widgets" (graphical objects) made by blending together widget sets provided by DEC and HP, and a window manager *mwm*. It seemed clear that both OPEN LOOK and Motif would be with us as "competing standards" for most of the 1990's.†

However, in 1993, some of the leading vendors got together and agreed on a new set of standards. The COSE group produced one, the Common Desktop Environment, that was based primarily on Motif. Thus it is likely that OPEN LOOK's importance will diminish over time. However, most COSE vendors do not plan to begin shipping the Common Desktop Environment until late in 1995, and in the meantime many people are using X and OPEN LOOK...

This book is about OPEN LOOK and X Windows. So we'll start this chapter by looking at a typical X display and consider some general features of the system. We'll also briefly compare a standard X application (written with the X Toolkit) to an OPEN LOOK applica-

† The OPEN LOOK specification, and all implementations, include these features not in Motif 1.1, many of which OSF is hoping to fit in to their next release: context-sensitive help, pinnable menus, and drag-and-drop. It is to be hoped that the OSF will provide these features in ways that are compatible with the widely-used "prior art" provided by the OPEN LOOK implementations features. OPEN LOOK also specifies the behavior of important clients such as the File Manager, Properties/Workspace manager, and others; these are described in upcoming chapters of this guide.

tion. Then we'll discuss what distinguishes the X Window System from other window systems. We'll also introduce some of the more important programs included in the standard distribution of X, and the *olwm* window manager shipped with OPEN LOOK. In the next chapter we'll show you how to start your X session and how to work in the OPEN LOOK environment.

1.1 Anatomy of an X Display

X is typically run on a workstation with a large screen. The X Window System also runs on PCs with 80386 CPUs and VGA graphics, on special X terminals, and on many larger systems. X allows you to work with multiple programs simultaneously, each in a separate *window*. The display in Figure 1-1 includes several windows.

The operations performed within a window can vary greatly, depending on the type of program running it. Certain windows accept input from the user: they may function as terminals, allow you to control a database, create graphics, etc. Other windows simply display information, such as the time of day, the system load average, a picture of the characters in a given font, etc.

The windows you will probably use most frequently are *terminal emulators*, windows that function as standard terminals. The terminal emulator included with the MIT release of X is called *xterm*; an OPEN LOOK standardized version of this program is called *olterm* on some systems. The terminal emulator used by default under OpenWindows is called *cmdtool* or *shelltool*. Figure 1-1 depicts two *xterm* windows and one *cmdtool* window. In a terminal emulator window, you can do anything you might do when logged in at the console or on a regular terminal: enter shell commands, run editing sessions, compile programs, etc.

Don't worry if you have already started up OPEN LOOK on your system and discovered that the screen looks nothing like this example—it won't!. We will show you how to customize your screen layout to make it look just the way you like. You will also find, if you are on a color display, that the frills around the edges of each window (called "window decorations") look more sculptured than the plain ones shown here. A nice feature of OPEN LOOK, unlike many other X11 GUIs, is that it automatically adapts to the kind of screen you are on. On a monochrome monitor, OPEN LOOK uses the "two-dimensional" look shown in most of these examples, which we've used because it prints more cleanly in the book. When running on a color or grayscale monitor, OPEN LOOK attempts to create a three-dimensional appearance[†], which is more attractive than the look provided by some older window managers. See the page of Color Plates for some examples of OPEN LOOK in color. If you're using such a monitor, you'll probably notice that window frames, various command buttons, icons, etc., appear to be raised to varying heights above screen level. This illusion is created by subtle shading and gives many display features a "beveled" look, similar to the beveled style of some mirrors.[‡]

† Purists will object to this use of "three dimensional", since it hides an entire discipline of computer graphics work in the presentation of three-dimensional images. The rest of us will content ourselves with its use.

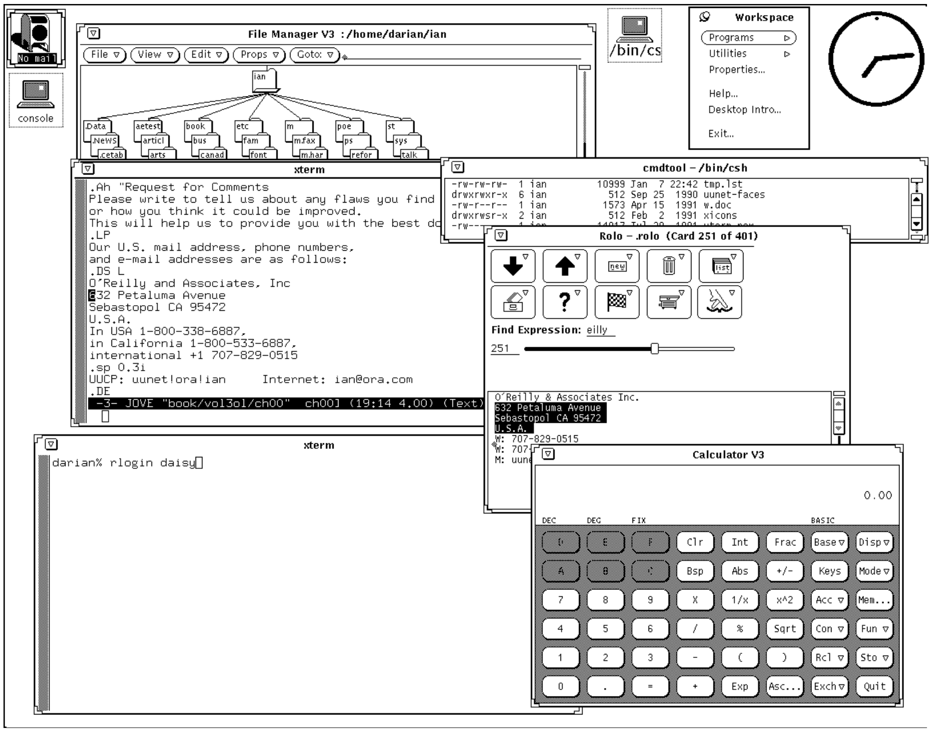

Figure 1-1. X display with numerous windows and an icon

The display in Figure 1-1 also includes other application windows: a clock, a mailing list program *xrolo*, and a calculator *calctool*. OPEN LOOK and X provide many such small utility programs—analogous to the so-called "desk accessories" of the Macintosh environment—intended to make your work easier.

The blank area that fills the unused portion(s) of the screen is conventionally called the *root* (or *background*) window; OPEN LOOK calls this area the *Workspace*. Application windows are displayed on top of this root window. X considers windows to be related to one another in a hierarchy, similar to a family tree. The root window is the root or origin within this hierarchy and is considered to be the *parent* of application windows displayed on it. Conversely, these application windows are called *children* of the root window. In Figure 1-1, the *xterm*, *cmdtool*, clock, *xrolo*, and *xcalc* windows are children of the root window.

The window hierarchy is actually much more complicated than this "two generation" model suggests. Various *parts* of application windows are windows in their own right. For example, many applications provide menus to assist the user. Technically speaking, these menus are separate windows. Knowledge of the complexity of the window hierarchy (and

‡ A few people find this "three-dimensional" look distracting, so of course OPEN LOOK provides a way to turn it off, as we'll see later. Other Graphical User Interfaces that provide a 3D look, like Motif, generally do not give you this option.

the composite parts of an application) is irrelevant at this stage, but will become important when we discuss how to tailor an application better to suit your needs.

One of the strengths of a window system such as X is that you can have several processes running in (and even writing output to) different windows simultaneously. For example, in Figure 1-1, the user is logging in to a remote system in one *xterm* window and editing a text file in another terminal emulator window. And he's fetching an address from a "Rolodex" emulator in another for use in the text file. (As we'll see in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, you can also cut and paste text between windows.) The only limitation on this multiwindow activity is that you can only be typing input into one window at a time.

Another strength of X is that it allows you to run programs on machines connected by a network. You can run a process on a remote machine while displaying the results on your own screen. You might want to access a remote machine for any number of reasons, to use a program or access information not available on your local system; to distribute the work load, etc. We'll discuss X's networking capabilities in more detail in the "X Architecture Overview" on page 20.

Now let's take another look at our sample display in Figure 1-1. Notice that some of the windows overlap each other. Windows often overlap much like sheets of paper on your desk or a stack of cards. This overlapping does not interfere with the process run in each window. However, to really take advantage of windowing, you need to be able to move and modify the windows on your display. For example, if you want to work in a window that is partially covered by another window, you need to be able to raise the obscured window to the top of the window stack.

Window management functions are provided by a program called a *window manager*. The window manager controls the general operation of the window system, allowing you to change the size and position of windows on the display. You can reshuffle windows in a window stack, make windows larger or smaller, move them to other locations on the screen, etc. The window manager provides part of the user interface—to a large extent it controls the overall "look and feel" of the window system.

The window manager provided with OPEN LOOK is called *olwm*, the *O*pen *L*ook *w*indow *m*anager. There are two implementations of *olwm*, one from Sun and one from AT&T. At this stage it does not matter which you are using (only when we come to customize the window manager behavior will the differences become important). A nice feature of *olwm* is the "frame" it places around all main windows on the display. Each top-level application window in the illustration is surrounded by this frame. As we'll see, you can manage the window by clicking a mouse[†] or other pointing device on various parts of the window frame.

† Today, the mouse is an integral part of the computer community, but it was not always so. The first computer mouse is believed to have been invented in 1964 by Doug Engelbart at SRI, near Stanford, California. He writes (in A. Goldberg (Ed.), *A History of Personal Workstations*, p. 196): "No one is quite sure why it got named a 'mouse', or who first started using that name. None of us would have thought that the name would have stayed with it out into the world, but the thing that none of us would have believed either was how long it would take for it to find its way out there."

The *olwm* window frame is a composite of several parts, the most prominent of which are shown in Figure 1-2. The top edge of the frame is wider than the other three edges and fea-

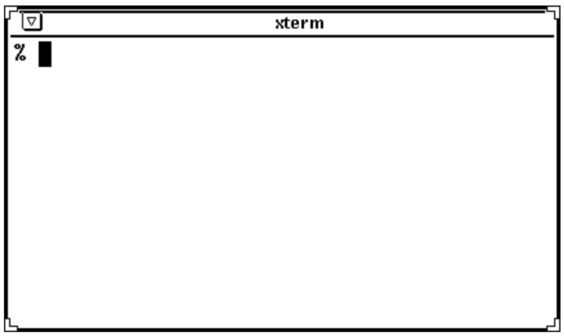

Figure 1-2. olwm frames each window on the display

tures most of the window management tools. This wide horizontal bar spanning the top of the window is called a *header*, or in standard X11 terminology, a *titlebar*. Most X11 window managers provide titlebars. The large central portion of this top edge is called the *title area* because it contains a text description of the window. (Generally this is the application name, but as we'll see later you can often specify an alternate title.) The titlebar also features a small menu mark—a box with a triangle—that we'll discuss in Chapter 6, *Using the OPEN LOOK Window Manager*. We'll also see how to use the parts of the frame to resize a window and to raise it to the top of the window stack.

olwm is intended to be used with applications built using the OPEN LOOK graphical user interface. For those not using OPEN LOOK, there are several other window managers available. In the standard distribution of X from MIT (as of X11 Release 4 and 5), the official window manager is *twm*. *twm* originally stood for "Tom's window manager," in honor of its developer, Tom LaStrange. However, it has since been worked on by many people and subsequently renamed the "tab window manager." The *twm* window manager provides a different "look and feel" than *olwm*. Rather than framing application windows, *twm* simply provides each window with a titlebar, different from the *olwm* titlebar in style, but offering similar window management functions. Other window managers exist, each with its own distinctive look and feel and its own set of features. The best known commercial offering besides *olwm* is *mwm*, which is normally used with Motif applications. However, you should note that most Motif applications will run with an OPEN LOOK window manager, and *vice versa.*

Aesthetics notwithstanding, one of the primary advantages that *olwm* shares with *mwm*, in contrast to some other window managers such as *twm*, is inherent in the nature of a frame: it provides window management tools on four sides of the window. *twm*'s titlebar is a useful window management tool, but a titlebar is often covered by other windows in the stack. In most cases at least a part of a window's frame will be visible—and thus accessible to *olwm* users but not to *twm* users.

Also pictured in Figure 1-1 are several *icon*s. An icon is a small symbol that represents a window in an inactive state. The window manager allows you to convert windows to icons and icons back to windows. You may convert a window to an icon to save space on the display, or to prevent input to that window. (Some programs, such as Sun's SunPC MS-DOS emulator, become "dormant" when iconified, and stop using CPU time.) Each icon has a label, generally the name of the program that created the window. The icons in Figure 1-1 represent another *xterm* window, a special system console window, and the file manager tool that we'll meet in Chapter 2, *Working in the OPEN LOOK Environment*. Icons can be moved around on the display, just like windows.

The contents of a window are not visible when the window has been converted to an icon, but they are not lost. In fact, a client continues to run when its window has been iconified; if you iconify a terminal emulator client such as *xterm*, any programs running in the shell will also continue.

All OPEN LOOK displays require you to have some sort of pointer, usually a three-button mouse, with which you communicate information to the system. As you slide the pointer around on your desktop, a cursor symbol on the display follows the pointer's movement. For our purposes, we will refer to both the pointing device (e.g., a mouse) and the symbol that represents its location on the screen as pointers. Depending on where the pointer is on the screen (in an *xterm* window, in another application window, on the root window, etc.), it is represented by a variety of cursor symbols. If the pointer is positioned on the root window, it is generally represented by an arrow-shaped cursor, as in Figure 1-1. If the pointer is in an *xterm* window, it looks like an uppercase "I" and is commonly called an *I-beam cursor*.†

The cursor often changes shape as you move the pointer into different windows. A complete list of standard X and OPEN LOOK cursors is shown in Appendix D, *Standard Cursors - X11 and OPEN LOOK*. Some common MIT cursor shapes, as well as two OPEN LOOK-specific cursors, are shown in Figure 1-3. As we'll see later, some applications allow you to select the cursor to use.

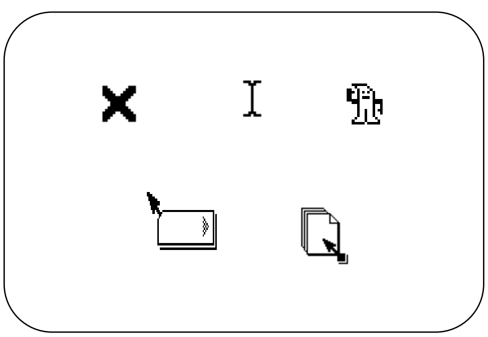

Figure 1-3. Three X11 standard cursors and two OPEN LOOK-specific cursors

† Even though the actual image on the screen is called a cursor, throughout this guide we refer to "moving the pointer" to avoid confusion with the standard text cursor that can appear in a terminal emulator window.

You use the pointer to manage windows and icons, to make selections in menus, and to select the window in which you want to input. You can't type in a terminal window until you select that window using the pointer. Directing input to a particular window is called *focusing*. When a window "has the input focus," the titlebar and the text cursor (if any) are highlighted. The window to which input is directed is often called the *active* window. Most window managers require you to select the active window in one of two ways: either by moving the pointer so that it rests within the desired window or by clicking the pointer on the window. By default, the OPEN LOOK Window Manager requires you to click the pointer on the window to which you want to direct input. This focusing style is commonly referred to as "click-to-type" or "explicit focus." In this mode, the active window is indicated by darkening its titlebar, as shown in Figure 1-4.

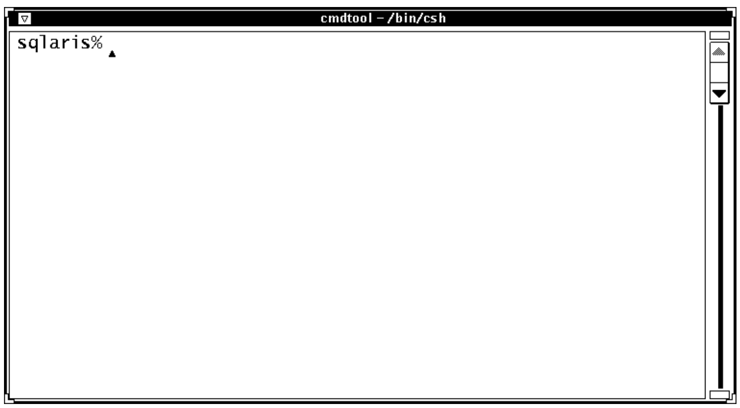

Figure 1-4. Focus on a terminal emulator window (click-to-type focus)

As we'll see in Chapter 13, *Customizing olwm*, *olwm* can be customized to allow you to direct input focus simply by moving the pointer. This alternate focusing style is commonly referred to as "real-estate-driven" (or "pointer focus"). In this mode, the active window is highlighted simply by drawing a line along the bottom of its titlebar, as shown in Figure 1-5.

The MIT *twm* window manager normally uses the real-estate-driven style: you direct input focus by moving the pointer into the desired window and leaving it there. As long as the pointer remains within the window's borders, the keystrokes you type will appear in that window (when the application accepts text input) or will somehow affect its operation (perhaps serve as commands). If you accidentally knock the pointer outside the window's borders, the keystrokes you type will not appear in that window or affect its operation. If you inadvertently move the pointer into another window, that window becomes the focus window. If you move the pointer onto the root window, the keystrokes are lost—no application will interpret them.

As mentioned, the OPEN LOOK window manager uses the click-to-type focusing style by default: you must click the pointer on a window to focus input on that window. When you begin using X with *olwm*, you'll need to select the window to receive input by placing the

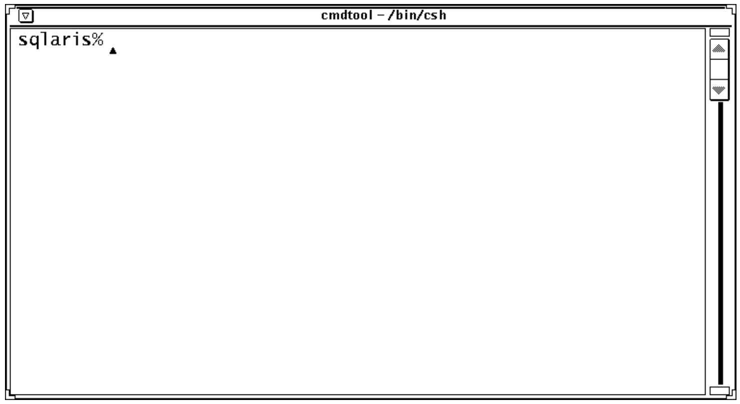

Figure 1-5. Focus on a terminal emulator window (pointer focus)

pointer anywhere within the window and clicking the button called SELECT, normally the leftmost button. †

Once you select the focus window in this way, all input is directed to that window until you move the pointer and deliberately click on another window. If you click SELECT on the root, then no window has the input focus until you click SELECT on the titlebar of a window.

One of OPEN LOOK's strengths is that it allows you to choose the focus policy. This flexibility makes *olwm* a desirable choice for users with a variety of needs and work habits. As you might imagine, both focusing policies have their advantages. Click-to-type focus requires a little more work than pointer focus. (It's simpler to move the pointer than to move and click.) On the other hand, click-to-type focus is more precise—you can't inadvertently change the focus by moving the pointer.

We find click-to-type focus somewhat laborious. However, a touch typist, who is not inclined to look at the screen, might consider pointer focus too risky. It's possible to knock the pointer out of a window and type a large amount of text into the wrong window before noticing. Another disadvantage of pointer focus is that it sometimes takes a moment for the input focus to catch up with the pointer, especially on slower machines. If you type right away, some keystrokes may end up in the window you left rather than in the new window. This problem is not unique to *olwm*, but rather is a side effect of the additional overhead caused by the generality (specified in a document called the ICCCM; see Volume Zero, *X Protocol Reference Manual*) involved in complex window managers, and can affect any ICCCM-compliant window manager such as *olwm*, *mwm*, or *twm*. Since you can change the focus policy rather simply, even while the OpenWindows version of *olwm* is run-

† The middle button is called ADJUST, and the right button is called MENU. To remember which button is which, think of "SAM" for (in order) SELECT, ADJUST, and MENU. We'll see how to use all these buttons as we progress. If you are left handed and want to reverse the position of these buttons, see the last section of Chapter 14, *Customization Clients*.

ning† you might want to experiment with both methods. For now, we'll assume you're using the default click-to-type focus.

The most important thing to recognize is that focusing on the proper window is integral to working with an application running with a window system. If something doesn't work the way you expect, make sure that the focus is directed to the correct window. After you use X for a while, awareness of input focus will come naturally.

The pointer is also often used to display menus. Most X programs, notably *olwm*, *shelltool* and *xterm*, have menus that are displayed by keystrokes and/or pointer button motions. *olwm* provides two default menus—the *Window Menu* and the *Workspace Menu*—each representing a different menu "style."

The *Window Menu* is a "pull-down" menu that can be displayed on any window by placing the pointer on the titlebar on top of the frame and either clicking or pressing and holding down the MENU pointer button. Roughly defined, a pull-down menu is accessed from a graphical element that is always displayed, such as a command button, a menu bar, or a titlebar. Figure 1-6 shows a terminal emulator window (in this case *cmdtool*) with the *Window Menu*) displayed by clicking the MENU pointer button in the menu button on the frame.

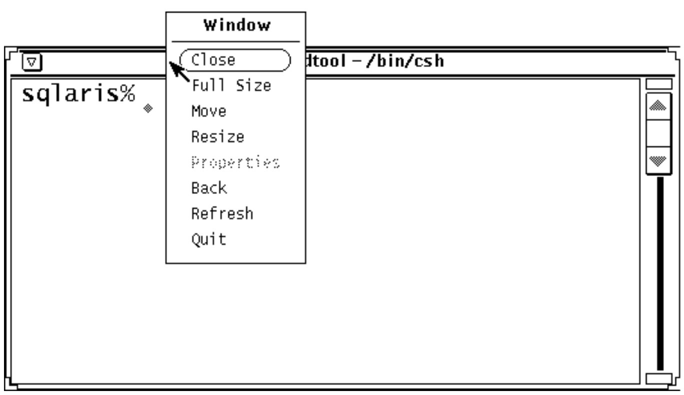

Figure 1-6. A pull-down menu: olwm's Window Menu

As you might infer from some of the menu items, you can use the *Window Menu* to hide, resize, and otherwise manage the window on which it is displayed. When you display the *Window Menu* by clicking the MENU pointer button (as opposed to pressing and holding it down), the default item (the one you would get just by clicking SELECT on the triangle mark) is surrounded by an oval box. In this case, the first available selection is *Close*, used to change a window into an icon. Of course, the first item in the *Window Menu* of an icon

† You can change it at start-up time by specifying **-follow** on the *olwm* command line, by setting the X Resource OpenWindows.SetInput: to followmouse, or change it at will by putting an item with a call to the FLIPFOCUS built-in function in your *.openwin-menu.* file, and clicking on it. All these methods are described in more detail in Chapter 13, *Customizing olwm*.

is *Open* which converts the icon into a window. The third item, *Properties*, is not available yet; it is meant to provide access to a set of Window Properties. The fact that it is not selectable is indicated by the fact that it appears in a lighter (or stippled) typeface. The second item, *Full Size*, and other aspects of the *Window Menu*, are discussed in detail in Chapter 6, *Using the OPEN LOOK Window Manager*.

OPEN LOOK also supports "pop-up" menus, which are displayed at the current pointer position (many X clients also use pop-up menus). In addition to keyboard keys and pointer button motions, the location of the pointer plays a role in displaying menus. For example, *xterm* menus can only be displayed when the pointer is within an *xterm* window. Figure 1-6 shows the *olwm Root Menu*, a pop-up menu that is generally displayed by placing the pointer on the root window and pressing and holding down the MENU pointer button.

In Figure 1-6, the arrow next to the menu title represents the pointer. As you drag the pointer down the menu, each of the menu selections is highlighted (OPEN LOOK programs generally highlight an item by placing an oval box of reverse video around it). When you release the pointer button, the highlighted menu item is selected. If you drag the pointer right off an OPEN LOOK menu without selecting anything, the menu stays up until either you release the pointer button (in which case nothing is selected and the menu pops down), or you move the pointer back into the menu and select something.

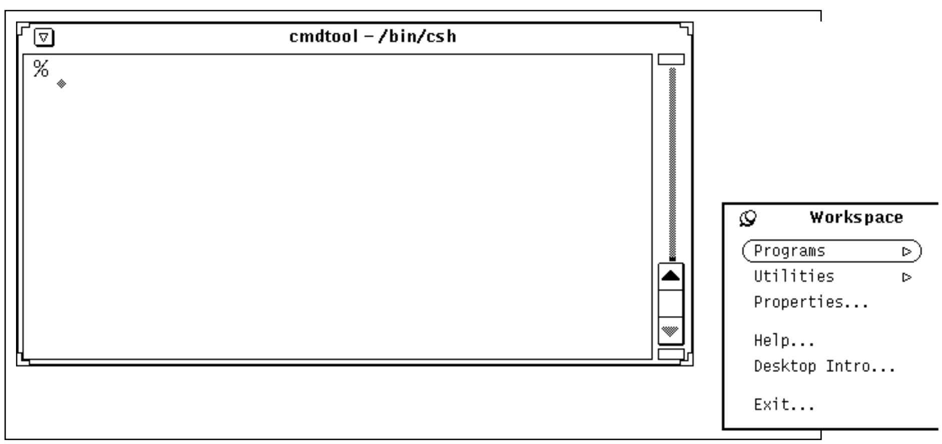

Figure 1-7. A pop-up menu: olwm's Workspace (Root) Menu

The *Workspace* or *Root Menu* (see Figure 1-7) provides commands that can be thought of as affecting the entire display (as opposed to a single window). For example, the first menu item, *Command Tool* (on OpenWindows) creates a new *cmdtool* window on the local machine and display.

Though *olwm*'s menus are useful, you'll probably find that you perform many window management functions simply by using the pointer on the window frame. In Chapter 6, *Using the OPEN LOOK Window Manager*, we'll describe several of these functions. Keep in mind, however, that both of the menus can be useful in certain circumstances. For instance, the *Window Menu* may be useful when parts of the window frame are obscured by another window. The *Workspace Menu* can be customized to execute system commands, such as the terminal emulator command initialized by the *Command Tool*'s item under *Programs*. It's fairly simple to add items to the OPEN LOOK root menu or *Workspace Menu* for the applications you use regularly. Just to show you the range of flexibility, Table 1-1 shows an alternative workspace menu that we use.

Table 1-1. Alternate Workspace Menu

Name	Function
Local Windows	Create local terminal emulator windows
LANlogins	Terminal emulator windows on our Local Area Network
Internet	Terminal emulator windows on Internet hosts
Window Programs	One of several dozen X- and NeWS- based applications
OpenWin Demos	A link to the standard OpenWindows demo menu file
Lock Screen	A standard X11 screen lock program
Terminations	Shutdown X or even shutdown my workstation

Chapter 13, *Customizing olwm*, describes how to add menu items and other modifications.

As we'll see in Chapter 8, *Other Standard Clients*, a few non-OPEN LOOK programs provide "jump-up" menus that you display simply by placing the pointer on a particular part of the window, e.g., a horizontal menu bar across the top.

A final note about the X display: in X, the terms *display* and *screen* are not equivalent. A display may consist of more than one *screen*. This feature might be implemented in several ways. There might be two physical monitors, linked to form a single display, as shown in Figure 1-8. Alternatively, two screens might be defined as different ways of using the same physical monitor. For example, on certain older Sun Workstations (any Sun with a "cgfour" color display), screen 0 is normally color, and screen 1 is black and white.†

† With OpenWindows, you must specify the screens to use and their order with the `-device` command line option; the default is a single screen in color. See the example in the OpenWindows section of Chapter 2, *Working in the OPEN LOOK Environment*, or the *openwin* manual page in Part III of this guide for details. The MIT "Xsun" server uses the same technique on a cgfour, but automatically uses both screens. With either server, the OpenWindows version of *olwm* automatically finds all the screens that are in use.

Each screen is the size of the monitor; you can only view one screen at a time. In practice, the two screens seem to be side by side: you can "scroll" between them by moving the pointer off either horizontal edge of the screen. By default, windows are always placed on screen 0 but you can place a client window on screen 1 by specifying the screen number in the `-display` option when starting the client. (See Chapter 2, *Working in the OPEN LOOK Environment*, for instructions on using the `-display` option.)

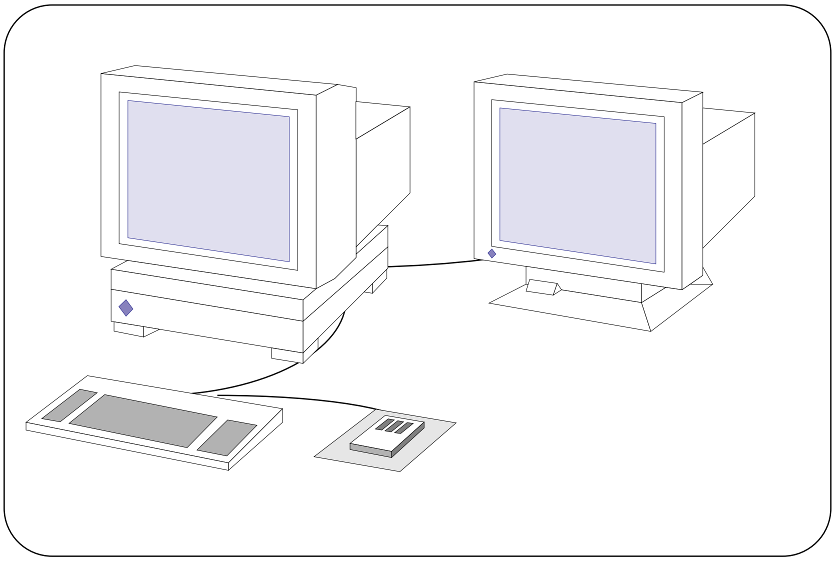

Figure 1-8. A display made up of two physical screens

1.2 Standard X Clients, OPEN LOOK Clients and Motif Clients

The window manager running on a display helps determine the look and feel of an application. The *olwm* window manager frames each window on the display and allows you to manage a variety of application windows using the same mechanisms.

However, the look and feel of an application is not entirely determined by the window manager. The programming routines used to create the application also distinguish it. The *olwm* and *cmdtool/shelltool* programs were written with an OPEN LOOK-specific toolkit, while the *xterm* program was been written using "the X Toolkit", developed at MIT.

The X Toolkit is a collective name for two subroutine libraries designed to simplify the development of X Window System applications: the X Toolkit (Xt) Intrinsics and the Athena widget set (Xaw). The Xt library consists of low-level C programming routines for building and using *widgets*, which are pre-defined user interface components or objects. Typical widgets create graphical features such as menus, command buttons, dialog boxes, and scrollbars. Widgets make it easier to create realistic applications. A common widget set also ensures a consistent user interface between applications.

As mentioned, most of the "standard" MIT X applications are written using the X Toolkit, and have the user interface provided by the Athena Widget set.

One of the first implementations of OPEN LOOK for X Windows is AT&T's OPEN LOOK Intrinsics Toolkit (OLIT). As the name implies, OLIT is based on the Xt Intrinsics, and consists primarily of a widget set that implements the OPEN LOOK GUI. An application coded using the OLIT Toolkit, or any other OPEN LOOK toolkit, has a particular look and feel that is prescribed in the *OPEN LOOK Style Guide* and illustrated by most of the pictures in this book.

Another OPEN LOOK implementation for X is the XView toolkit, designed by Sun to allow the thousands of SunView applications to migrate to OPEN LOOK and The X Window System. Its programmer interface is similar to that of the older proprietary SunView window system, but it is an OPEN LOOK toolkit that works with X.

A third OPEN LOOK toolkit—one that is being phased out in favor of the others—is "The NeWS Toolkit" (TNT), also from Sun. This toolkit is neither Xt- nor X11-based, but uses the alternate NeWS protocol to communicate with the display server. It is included in Sun's OpenWindows package; applications written using TNT should behave much the same as those developed using either of the other OPEN LOOK toolkits.

The Open Software Foundation has also developed an Intrinsics-based widget set – or more precisely, a specification for a GUI—called Motif. There is only one sanctioned implementation of the Motif specification, and if you want it, you must license it from the Open Software Foundation. OPEN LOOK and Motif are the prime contenders to establish a graphical user interface standard in the market. In fact, they *are* the two standard GUIs in the workstation market. We will discuss OPEN LOOK clients throughout most of this book. Since you may occasionally need to use Motif-based clients, we show one sample Motif program later in this chapter, and summarize the important aspects of using Motif programs in Appendix M, *OPEN LOOK and Motif*.

Some of the clients discussed in this guide are standard X clients shipped by MIT, while others are part of the commercial OPEN LOOK offerings. Most of the MIT clients have been built with the X Toolkit and illustrate the use of many of the Athena widgets. When you run these standard clients (or any Motif clients) with the *olwm* window manager, your environment is something of a hybrid—neither a vanilla X nor a pure OPEN LOOK environment.

A standard X Toolkit client running with *olwm* is different from a true OPEN LOOK application, coded using an OPEN LOOK Toolkit. At first look, they may seem similar—when *olwm* is running, all clients on the display are framed in the same way. In addition, certain graphical features provided by the OPEN LOOK specification are also provided, albeit with variations, by the Athena widgets. However, other features are unique to OPEN LOOK.

Without dissecting every component or closely examining how it functions, let's briefly compare a standard X application to an OPEN LOOK application, highlighting some of the major differences (primarily in appearance). Many features of OPEN LOOK and standard X applications also *operate* differently. We'll examine the functionality of various OPEN LOOK and Athena widgets in more detail in Chapter 8, *Other Standard Clients*.

For the comparison, we'll use an OPEN LOOK client program called *textedit*, the OpenWindows standard window-based text editor. *textedit* (see Figure 1-9) demonstrates many of the features of OPEN LOOK as a GUI. It also demonstrates most of the OPEN LOOK-specific editing actions: editing text in this program is the same as editing commands in *cmdtool*'s history mechanism, for example. This aspect of *textedit* is described more fully in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

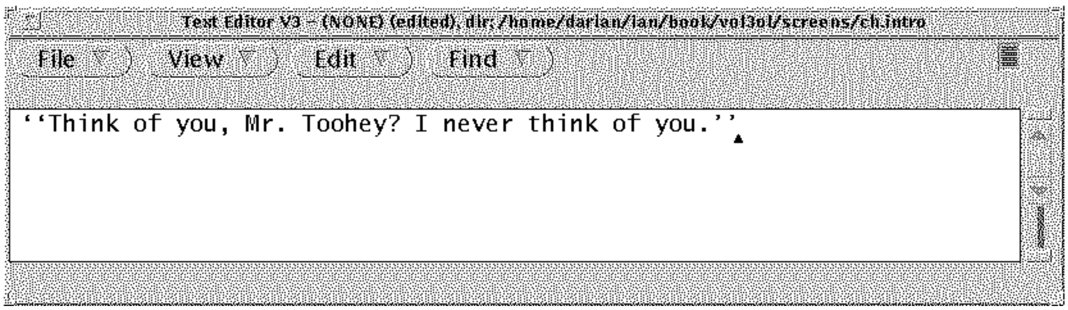

Figure 1-9. An OPEN LOOK application: textedit

The standard X client we're using is *xclipboard* (Figure 1-10), which you use in concert with *xterm*'s "cut and paste" facility, described in Appendix A, *The xterm/olterm Terminal Emulator*. The *xclipboard* client provides a window in which you can paste multiple text selections and from which you can copy text selections to other windows. Similar to the clipboard feature of the Macintosh operating system, *xclipboard* is basically a storehouse for text you may want to paste into other windows.

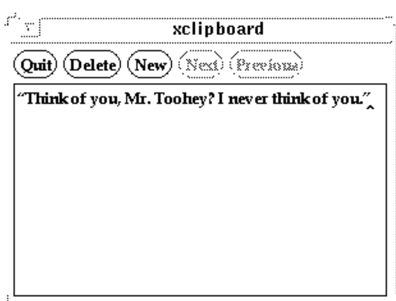

Figure 1-10. An MIT X application: xclipboard

A Motif application displayed in Figure 1-11 is *mre*, the Motif-based Resource Editor (we'll talk about Resources themselves later). *mre* is a demonstration program that is normally shipped with Motif, but isn't part of the official Motif standard. In fact, the Motif standard doesn't document any standard clients, so your friend's Motif system down the hall may or may not include it.

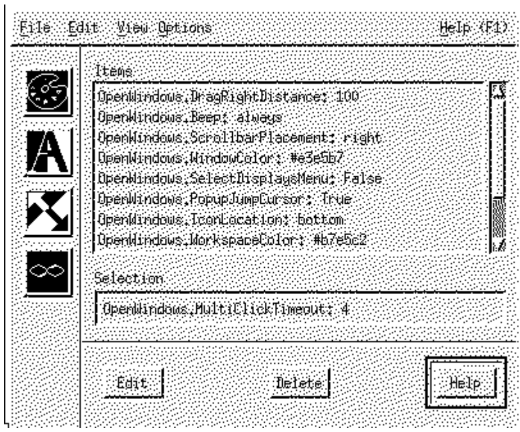

Figure 1-11. A Motif application: mre

The most striking difference between the first two clients is simply the amount of detail. Like the OPEN LOOK window manager, the individual OPEN LOOK controls create a "three-dimensional" appearance, at least on a color or grayscale monitor†. The push buttons are shaded to suggest that they are raised above the level of the application window. The menu bar is shaded to appear raised. The scrollbars have clearly distinguishable components, all of which are shaded and contoured to maintain the 3-D impression. For a better look at this effect, see the page of color plates; it has been reduced to monochrome for printing here. The Motif window, too, uses a bevelled or 3-D impression.

By contrast, the *xclipboard* window seems almost like a preliminary sketch of an application. It is basically flat. The text window, command buttons, and scrollbars are rendered in simple lines, without contouring, and with virtually no shading (though a portion of each scrollbar is shaded).

Each application has a series of buttons along the top. In *xclipboard*, each button simply does one action. In an OPEN LOOK or Motif application, however, these buttons are usually pull-down menus. OPEN LOOK clearly indicates these menus by the downwards-pointing triangle. *File, View*, and *Edit* are common on OPEN LOOK applications. You can activate one of these menus by placing the pointer on it and clicking (or pressing and holding down) the MENU (right) pointer button. Alternately, you can select the default action from that menu, without making the menu appear, by clicking the SELECT (left) pointer button while the pointer is in the button. The *xclipboard* application doesn't provide any menus—it's a fairly simple program. However, some standard X clients (notably *xman* dis-

† Most of the screenshots of OPEN LOOK applications in this book are monochrome, and have a "two-dimensional" look. The *textedit* application was captured in grayscale to show the bevelling effects.

cussed in Chapter 8, *Other Standard Clients*) provide pull-down menus accessed by pressing and holding down a pointer button.

OPEN LOOK pull-down menus have a few advantages over pull-down menus provided by standard MIT X clients. While you must press and hold down a pointer button to display a menu provided by a standard X client, you can display an OPEN LOOK menu simply by clicking a pointer button—and the menu stays displayed until you click again. OPEN LOOK menu items can also be invoked in multiple ways (including pointer actions and keystrokes); the only way to invoke an item from a standard X menu is by dragging the pointer down the menu and releasing the button. The various ways you can work with an OPEN LOOK pull-down menu are described in Chapter 2, *Working in the OPEN LOOK Environment*.

One minor advantage of Motif applications is that their menu bar features *sliding menus*; if you hold the left button and slide the pointer across the menu bar, each menu will pop down without having to be clicked on.

Despite differences in general appearance and complexity, *textedit* and *xclipboard* have many analogous components. Both applications feature a subwindow containing text that can be edited.

Each application features buttons: push buttons in the *textedit* window; command buttons in the *xclipboard* window, drawn buttons in the *mre* window. From a user's viewpoint, push buttons and command buttons are functionally equivalent (though you can invoke a push button's function in more ways). OPEN LOOK and the Athena widget set simply identify them by different names. This Motif application, like some OPEN LOOK applications, features drawn or "glyph buttons," buttons decorated with a small picture instead of being labeled with a word or two of text.

All three application windows have a vertical scrollbar, used to display text that is currently outside the viewing window. (These scrollbars are only displayed when the text read into the window extends beyond the bounds of the viewing area. If the text only exceeds the viewing area in one direction—either horizontally or vertically—only one scrollbar will be displayed.) The Athena scrollbar is basically rectangular (actually one rectangle within another). The OPEN LOOK scrollbar is much more intuitive—it provides the visual analogy of an elevator used in a multi-story building. Notice the arrows on either end of the indicator mark in the middle, for instance. These arrows are the hallmark of an OPEN LOOK scrollbar and can help you readily identify an OPEN LOOK application. The arrows also provide functionality not duplicated by the Athena scrollbar. Similarly, the Motif scrollbar can be used to recognize Motif applications.

In general, once you've mastered the basics of working with OPEN LOOK client programs running under the *olwm* window manager, you should have no problem making use of any additional features provided by commercial applications built with OPEN LOOK or any similar GUI.

1.3 X Architecture Overview

Before X11 became popular, most window systems were *kernel-based* and *host-based*: that is, they were closely tied to the operating system itself and could only run on a discrete system, such as a single workstation. X, by contrast, is a *user-mode*, *network-based* window system; it is not part of any operating system but instead is composed entirely of user-level programs.

The architecture of the X Window System is based on what is known as a *client-server* model. The system is divided into two distinct parts: *display servers* that provide display capabilities and keep track of user input and *clients*, application programs that perform specific tasks.

In a sense, the server acts as intermediary between client application programs, and the local display hardware (one or perhaps multiple screens) and input devices (generally a keyboard and pointer). When you enter input using the keyboard or a pointing device, the server conveys the input to the relevant client application. Likewise, the client programs make requests (for information, processes, etc.) that are communicated to the hardware display by the server. For example, a client may request that a window be moved or that text be displayed in the window.

This division within the X architecture allows the clients and the display server either to work together on the same machine or to reside on different machines (possibly of different types, with different operating systems, etc.) that are connected by a network. For example, you might use a relatively low-powered PC or workstation as a display server to interact with clients that are running on a more powerful remote system. Even though the client program is actually running on the more powerful system, all user input and displayed output occur on the PC or workstation server and are communicated across the network using the X protocol. Figure 1-12 shows a diagram of such a network.

You might choose to run a client on a remote machine for any number of reasons. Generally, however, the remote machine offers some features unavailable on your local machine: a more efficient or powerful processor; a completely different architecture better suited to a particular task; different application software that is either licensed for that machine or only available on a particular architecture; file server capabilities (and perhaps large data files you'd rather not transfer over the network). X allows you to take advantage of these remote features and to see the results on your local terminal.

The distinction between clients and the server also allows for somewhat complicated display situations. For instance, you can access several machines simultaneously (this can greatly simplify the work of a system administrator). X also allows you to output to several displays simultaneously. This capability can be very helpful in educational situations; for example, a teacher could display instructional material to a group of students each using a graphics workstation or X terminal hooked up to a network.

There is another less obvious advantage to the client-server model: since the server is entirely responsible for interacting with the hardware, only the server program must be machine-specific. X client applications can easily be ported from system to system.

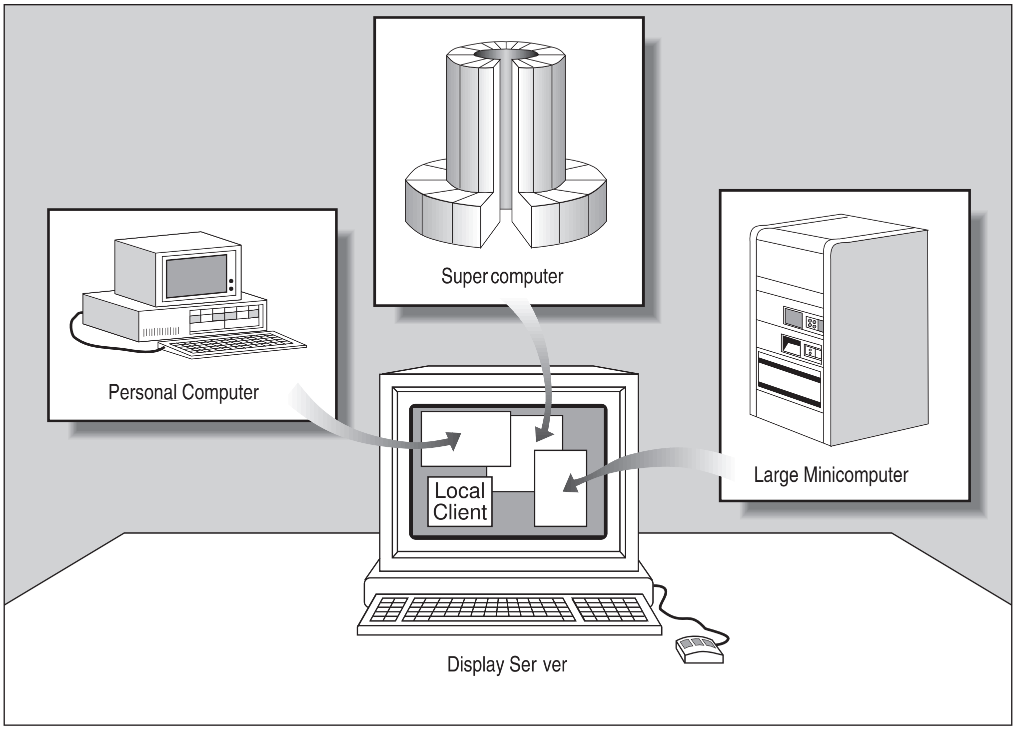

Figure 1-12. A sample X Window System configuration

1.4 The X Display Server

The X display server is a program that keeps track of all input from input devices, such as the keyboard and pointer, and conveys that input to the relevant client applications. The server also keeps track of output from any clients that are running and updates the display to reflect that output. Each physical display (which may be multiple screens) has only one server program.

User input and several other types of information pass from the server to a client in the form of *events*. An event is a packet of information that tells the client something it needs to act on, such as keyboard input. Moving the pointer or pressing a key, etc., causes *input* events to occur.

When a client program receives a meaningful event, it responds with a *request* to the server for some sort of action affecting the display. For instance, the client may request that a window be resized to particular dimensions. The server responds to requests by a client program by updating the appropriate window(s) on your display.

Servers are available for PCs with 80386 processors, for workstations, and even for special terminals (called X terminals), which may have the server downloaded from another machine or stored in "read-only memory" (ROM). Details on the variety of servers available is included in Volume Eight, *X Window System Administrator's Guide*.

The server and the client programs communicate using a rigidly—defined protocol, The X Protocol, described in Volume Zero, *X Protocol Reference Manual*. Some servers may recognize particular "extensions" to the base protocol.

The Sun OpenWindows server uses the standard X Protocol, but for increased flexibility OpenWindows Versions 2 through 3.1 also accept connections using another, lower overhead protocol called the NeWS (Network-extensible Window System) protocol. The NeWS language allows the application programmer to extend the environment by writing procedures in the PostScript language. NeWS clients will only work with a NeWS server, which means an OpenWindows server; they will be discussed in this book with the notation "OpenWindows Only".

Interestingly, some of the features that have become "X extensions" were first featured as standard parts of NeWS. These include the SHAPE extension (used to draw circular windows) and the DISPLAY POSTSCRIPT extension. Display Postscript (DPS for short) is less flexible than the NeWS protocol, but does give programmers who must use X-based toolkits the ability to use the PostScript language. DPS has the further advantage that DPS clients will work with a DPS server from any vendor. Neither OpenWindows 2 nor 3.1 used DPS; the NeWS protocol in the server is Sun's original display-based PostScript interpreter. However, Sun and Adobe (the creator of Display Postscript) signed a technology transfer agreement in late 1992 which will see the OpenWindows product switching over to Display PostScript soon †

1.5 Clients

As previously mentioned, a client is an application program. The OPEN LOOK Window Manager, *shelltool, xterm,* and *calctool* are client programs we've seen already. The standard release of X from MIT includes more than 50 client programs that perform a wide variety of tasks. Another few hundred client programs are available as "contributed" or "free software," and more are added each week. X allows you to run many clients simultaneously: each client displays in a separate window. For example, you could be editing a text file in one window, compiling a program source file in a second window, and reading your mail in a third, while displaying the system load average in a fourth window.

While X clients generally display their results and take input from a single display server, they may each be running on a different computer on the network. It is important to note that the same programs may not look and act the same on different servers: users can run different window managers on different servers; users can customize X clients differently on each server; and the display hardware on each server may be different.

Remember that the server conveys input from the various input devices to the appropriate client application; likewise, the client issues output in the form of requests to the server for certain actions affecting the display.

† This will happen by mid-1993?, probably with Release 3.2? of the OpenWindows system.

In addition to communicating with the server, a client sometimes needs to communicate with other clients. For example, a client may need to tell the window manager where to place its icon. Interclient communication is made easier by the use of *properties.* A property is a piece of information associated with a window or a font and stored in the server. Properties are used by clients to store information that other clients might need to know, such as the name of the application associated with a particular window. Storing properties in the server makes the information they contain accessible to all clients.

A typical use of properties in interclient communication involves how a client tells the window manager the name of the application associated with its window. By default, the application name corresponds to the client's name, but many clients allow you to specify an alternative name when you run the program. A window manager that provides a titlebar needs to know the application name to display in that area. The client's application name is stored in the server in the property called WM_NAME and is accessed by the window manager.

See 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.

! Note that OPEN LOOK has an additional, unrelated meaning for the term "properties", referring to "the properties of an application." This use of the term means "the user-controllable aspects of that program's behavior." This is exemplified by "property sheets" that let you edit the properties of an application; most OPEN LOOK applications include a Properties menu item, and the root menu has a like-named item that lets you edit global settings by adjusting X Resources. This is described in Chapter 6, *Using the OPEN LOOK Window Manager.*

We've already discussed one important client, the OPEN LOOK Window Manager. Several other frequently used client programs are discussed in the following sections.

1.5.1 The ShellTool terminal emulator

X11 itself is designed to support only bitmapped graphics displays. But there are many day-to-day operations that are best handled by a command-line interpreter like the UNIX shells. For this reason, one of the most important clients is a *terminal emulator*. The terminal emulator brings up a window that allows you to log in to a multiuser system and to run applications designed for use on a standard alphanumeric terminal. Anything you can do on a terminal, you can do in this window. If you are using Sun's OpenWindows package, the terminal emulator used in the default setup is called *shelltool* or *cmdtool* (the two names refer to a single program under two different modes).

Running multiple *cmdtool* processes is like working with multiple terminals. Since you can bring up more than one *cmdtool* window at a time, you can run several programs simultaneously. For example, you can have the system transfer files or process information in one window while you focus your attention on a text-editing session in another window. As you might imagine, having what are in effect multiple terminals can increase your productivity remarkably. See Chapter 5, *The cmdtool/shelltool Terminal Emulator*, for more information about *cmdtool/shelltool*.

1.5.2 The xterm Terminal Emulator

There are several terminal emulators besides *cmdtool*. Users of MIT and SVR4 X systems generally use one called *xterm*. *xterm* is perhaps the most widely available terminal emulator, and it is available under OpenWindows too. *xterm* emulates a character-based terminal or a vector-graphics terminal. The standard MIT X11 version of *xterm* emulates a DEC VT102 character terminal; the *xterm* shipped with AT&T-OL emulates an AT&T 6386 console in character mode. Both versions can also emulate a Tektronix® 4014 as their graphics terminal. For each *xterm* process, you can display both types of windows at the same time but only one is active (i.e., responding to input) at a time. This is one area where *xterm* would seem to excel over Sun's *cmdtool*, since the latter has no ability to emulate a Tektronix terminal.†

1.5.3 Other X Clients

The standard distribution of X from MIT includes more than 50 client applications; most of these are included in OpenWindows and AT&T-OL. The client you will probably use most frequently is a terminal emulator, either *shelltool* or *xterm*. We've grouped some of the other more useful applications as follows: the "Availability" column lists "Standard" (standard MIT clients), "Contributed" (free software, obtain from `Uunet` or MIT, see the *Preface*), "OPEN LOOK" for clients that should be in any OPEN LOOK system, or "OpenWindows" for clients that are only included with Sun's Open Windows.

† Since the ANSI and Tektronix terminals are so different, Sun's older SunView window system featured a separate program called *tektool*, for that purpose. However *tektool*, has not (yet) been brought forward to OPEN LOOK. If you are an OpenWindows user in need of Tektronix emulation, the SunView version is still included in SunOS 4.1, and it can be used under OpenWindows, subject to the limits laid out in Appendix L, *Running SunView Applications on OpenWindows*. Or, you could use *xterm* for the times when you need Tektronix mode.

These and other client applications are described in Chapters 5 through 9. In addition, a ref-

1

Table 1-2. An Applications Sampler

Name	Description	Availability
Desktop Manager		
filemgr, olfm	File manager (desktop manager)	any OPEN LOOK implementation
Desk accessories		
xbiff	Mail notification program	Standard
faces	Fancier, visual mail notifier	Contributed
xclock, oclock	Clock applications	Standard
clock	Fancier clock	OpenWindows
xcalc	Desktop calculator	Standard
xload	System load monitor	Standard
perfmeter	Fancier load monitor	OpenWindows and xview only
xman	Manual page browser	Standard
xrolo	XView Rolodex™ emulator.	Contributed
cm	Calendar Manager	OpenWindows
calentool	Calendar manager	Contributed
Display and keyboard preferences		
properties	Sets display, keyboard and color preferences; updates your resource file accordingly	Any OPEN LOOK implementation
xset	Sets display and keyboard preferences, such as bell volume, cursor acceleration, and screen saver operation	Standard.
xmodmap	Allows you to map keyboard keys and pointer buttons to particular functions	Standard
Text Editing utilities		

Table 1-2. An Applications Sampler

Name	Description	Availability
xedit	Simple text editor	Standard
textedit	Simple text editor	OpenWindows
Font utilities		
xlsfonts	Lists available fonts	Standard
xfd	Displays the characters in a single font	Standard
xfontsel	Allows you to display multiple fonts sequentially and select a font to be used by another application	Standard
Graphics utilities		
bitmap	Bitmap editor	Standard
iconedit	Another bitmap editor	OpenWindows
atobm, bmtoa	Convert ASCII characters to bitmaps and bitmaps to ASCII characters	Standard
pageview	Show PostScript output	OpenWindows
snapshot	Fancy screen capture program	OpenWindows
xloadimage, xv	Control, display and manipulate bitmap files in a variety of formats	Contributed
xfig, xpic	One of many drawing programs. Also touchup,.others	Contributed
Printing applications		
xwd, xwud	Dumps and re-displays the image of a window to a file	Standard
xpr	Translates an image file produced by *xwd* to PostScript® or another format, suitable for printing on a variety of printers	Standard
xdpr	Combines xwd, xpr and lp/lpr	Standard
Removing a window		

Table 1-2. An Applications Sampler

Name	Description	Availability
xkill	Terminates a client application	Standard
pam	Hides a dead window	OpenWindows only
Window and display information utilities		
xlsclients	Lists the clients running on the display	Standard
xdpyinfo	Lists general characteristics of the display	Standard
xwininfo	Lists general characteristics of a selected window	Standard
xprop	Lists the properties associated with a window	Standard
psps	Lists the PostScript (and other) processes (window programs) in the XNeWs server	OpenWindows only

erence page describing each client and listing its options appears in Part Three of this guide. As more commercial and user-contributed software is developed, many more specialized programs will become available. For example, at the time of this writing (mid-1992), the word processing user has the choice of many X- or OPEN LOOK-based word-processing programs, including Word Perfect, FrameMaker, Interleaf, IslandWrite, DECWrite, and others, with SoftQuad Author/Editor and ArborText Publisher for more involved electronic publishing projects. The desktop graphics market includes Arts & Letters, Corel Draw, Island Draw/Paint, and others. Just about any type of commercial application you could want is now available for The X Window System, and most of them are available for the OPEN LOOK GUI environment. A list of programs—both commercial and contributed—that use the OPEN LOOK GUI is published in the Usenet newsgroup *comp.windows.open-look*, and a recent version of this is reprinted in XXX NEW APPN.

1.5.4 Customizing Clients

Most X clients are designed to be customized by the user. A multitude of command-line options can be used to affect the appearance and operation of a single client process. A few of the more useful command-line options are introduced in Chapter 2, *Working in the OPEN LOOK Environment*. Chapter 11, *Command-line Options*, discusses several options in detail. Part Three of this guide includes a reference page for each client that details all valid options.

X also provides a somewhat more convenient (if complicated) way to customize the appearance and operation of client programs. Rather than specifying all characteristics using command-line options, default values for most options can be stored in a file (generally called *.Xresources* or *.Xdefaults*) in your home directory. Each default value is set using a variable called a *resource*; you can change the behavior or appearance of a program by changing the *value* associated with a resource variable.

Generally, these resource values are loaded into the server using a program called *xrdb* (*X resource database* manager). Then the values are accessed automatically when you run a client. Storing your preferences in the server with *xrdb* also allows you to run clients on multiple machines without maintaining an *.Xresources* file on each machine. Because editing these resources by hand can be tedious, OPEN LOOK specifies a program to manage the resources that affect global OPEN LOOK characteristics. On OpenWindows this program is called *props* (for "properties"); on AT&T-OL it is called *olwsm*, the "workspace manager." It can be started by name or from the root menu. We will see this program in detail in Chapter 6, *Using the OPEN LOOK Window Manager*.

In addition, many OPEN LOOK applications have a "properties" menu that will allow you to edit the resources (or other customization file) for that application. We'll see examples of these in various places.

There is a separate customization file for the *olwm* window manager (called *.openwin-menu* on OpenWindows or *.olprograms* in AT&T-OL) which is also kept in your home directory. On OpenWindows, editing the *.openwin-menu* file lets you modify the root menu (the *Workspace* menu). On AT&T-OL, you can only change the list of programs to be run from the *Programs* sub-window of the *Workspace menu*; you use the *properties* program to modify this list of programs. Unlike some other window managers, *olwm* does not let you modify the behavior of the window manager, nor the key and pointer button sequences used to invoke actions, by editing this file. Instead, you use "X Resources", most of which can be set by the *properties* program. Chapter 12, *Setting Resources*, has more information.

Client customization is introduced in Chapter 14, *Customization Clients*, and is the general topic of Part Two of this guide.