CHAPTER 3

Opening Additional Windows

The previous chapter showed you how to start up OpenWindows or The X Window System and use the basic features of X11 and OPEN LOOK, using the OPEN LOOK Window Manager to control windows. In this chapter we show you how to:

- Start additional clients
- Start those clients in convenient places on the display.
- Customize a client process using command line options.
- Specify alternate default characteristics for a client program using "resource" variables.
- Arrange for your own particular set of windows to be created each time you start the window system.

3.1 Starting Additional Clients

Now that you know the basics of managing windows, you can start other X clients just as you can start another instance of the terminal emulator. The following sections describe how to open more client windows, place them on the display in convenient positions, and take advantage of X11s networking capabilities by running clients on other machines.

To start a client, at the command line prompt in any terminal emulator window, type the name of the client followed by an ampersand (&) to make the client run in the background. For example, by typing:

```
% oclock &
```

you can create a new window displaying a clock. The clock will appear in a the next spot in a stack coming down from the upper-left corner of the screen. You can then drag the *oclock* window to a more convenient location—say the upper-right corner, as in

Figure 3-1. (Remember, to move a window, place the pointer on the titlebar, press the SELECT button, drag the window outline to the desired location, and release the button.)

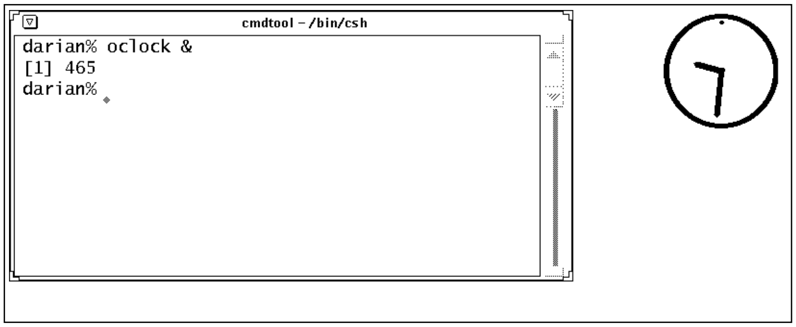

Figure 3-1. The oclock window

Though moving windows on the display is fairly simple, manually positioning every window each time you start using X is not particularly convenient. For most clients, X provides a way to specify a window's location (and also its size) automatically—using an option when you run the command. A window's size and position is referred to as its *geometry* and you set these attributes using the `-geometry` option. The use of this option is discussed in the section *Window Geometry: Specifying Size and Location* below

3.2 Command Line Options

Most X clients accept two powerful and extremely useful options: the `-geometry` option, which allows you to specify the size and location of a window on the screen; and the `-display` option, which allows you to specify on which screen a window should be created. Most commonly, you'd use the `-display` option to run a client on a remote machine and display the window 'locally', that is, on your display.

The next few sections illustrate some typical uses of these important options. In explaining how to use them, we introduce some new, perhaps somewhat involved concepts (such as the way distances can be measured on your screen). Bear with us. Though these two options are not part of the OPEN LOOK specification, we feel that you need to master the `-geometry` and `-display` options in order to begin to take advantage of the flexibility of OPEN LOOK and X.

Most clients will accept a `-help` option, which will print the commonly-used command-line options that the given client accepts. XView clients often print only the "generic" options in response to `-help`. More detail on command line options appears in Chapter 11, *Command-line Options*.

After explaining these options in detail, we'll briefly consider some of the characteristics you can specify using other common options.

3.2.1 Window Geometry: Specifying Size and Location

There is a command line option that is recognized by every imaginable X client to specify the size and placement of its base window on the screen. At first blush it looks a bit complicated. However, the OPEN LOOK Window Manager provides alternative methods of finding this information and saving it in a file. You can place your windows using the pointer, adjust their size using the pointer, then use the `Save Workspace` menu item. If you're willing to use that method, you can skip the remainder of this section until you need to know it.

The command line option to specify a window's size and location has the form:

`-geometry` ***`geometry`***

The `-geometry` option can be (and sometimes is) abbreviated to as much as `-g`, unless the client accepts another option that begins with "g." Applications developed using the XView toolkit cannot use -g because of other options†. Specifying `-geometry` is always safe! Particularly in shell scripts, you should always use the long form, so that the script won't stop working when a new version of a program is installed; using the long forms also makes the scripts more readable.

The parameter to the geometry option (***`geometry`***), referred to as the "standard geometry string," has four numerical components, two specifying the window's dimensions and two specifying its location. The standard geometry string has the syntax:

`width`**`x`**`height±xoff±yoff`

Obviously, the first half of the string provides the ***`width`*** and ***`height`*** of the window. Many application windows are measured in pixels. However, application developers are encouraged to use units that are meaningful in terms of the application. For example, *xterm*'s dimensions are measured in columns and rows of font characters. More precisely, an *xterm* window is some number of characters wide by some number of lines high (80 characters wide by 24 lines high by default). *Cmdtool/shelltool*'s `geometry` dimensions are measured in pixels; use the alternate `-Ww` *x* and `-Wh` *x* to specify the size in columns and lines for *cmdtool* or any XView client whose base window is a text window.

The second half of the geometry string gives the location of the window relative to the horizontal and vertical edges of the screen. Imagine the screen to be a grid (where the upper-left corner is 0,0), ***`xoff`*** (x offset) and ***`yoff`*** (y offset) represent the x and y coordinates at which the window should be displayed. The x and y offsets are also measured in pixels.

Many users may not be accustomed to thinking in terms of pixels. What exactly is a pixel? A pixel is the smallest element of a display surface that can be addressed by a program. You can think of a pixel as one of the tiny dots that make up a graphic image, such as that displayed by a terminal, an X display, or even a home television set.

Since a pixel is so tiny, gauging sizes and distances in pixels will take some practice. For instance, what are the dimensions of your screen in pixels? Your workstation or X terminal

† If you *like* short forms, you can use **-WG** with XView clients; it takes the same geometry specification as does -geometry.

documentation may provide this information, or you may have to experiment by placing windows in order to figure out the approximate dimensions.

Keep in mind that monitors can vary substantially. The Sun 19-inch monitor usually has dimensions of 1152 x 900 pixels (these figures also comprise what is known as the screen's *resolution*). Since the horizontal and vertical dimensions of the viewing area are approximately 13.75 x 10.75 inches, each inch is equivalent to approximately 84 pixels (sometimes called dots) in both dimensions. By contrast, the 16-inch monitor on the Sony NEWS workstation has dimensions of 1280 x 1024 pixels. The actual viewing area is approximately 13 inches wide by 10 inches high. This translates to 98.5 dots per inch (dpi) horizontally and 102.4 dpi vertically, or roughly 100dpi.

Most X servers on PC-class machines require the use of a VGA monitor board. A VGA display has a minimum resolution of 640x480, and the 13-inch monitor we measured has a viewing area about 9-3/4 x 6-6/8 inches, giving a resolution of 65.5 x 72.5 dots per inch. Since most "VGA compatible" screen adapters also support other higher resolutions, *there is no single resolution standard for PC-class machines with VGA*. The same monitor in an 800x600 mode would have a resolution of 82 x 90.5 dots per inch. And at "Super VGA" 1024x768, the resolution would be a respectable 105 x 116 dots per inch.

What are the implications of such hardware differences in specifying client geometry? The size and location of client windows is related to the size and resolution of your screen. For example, if you specify a window with dimensions of 125 x 125 pixels, it will appear somewhat larger on the Sun monitor than on the Sony or the Super VGA.

So how do we use the geometry option to specify a window's size and location? First, be aware that you can specify any or all elements of the geometry string. Incomplete geometry specifications are compared to the application's default settings and missing elements are supplied by these values. All client windows have a default size. For example, if you run an *xterm* window with the geometry option and specify a location but no dimensions, the application default of 80 characters by 24 lines is used.

If you don't specify the x and y coordinates at which to place a client window, the client may provide a default location; if the application doesn't provide a default, the *olwm* window manager will automatically position the window in a "stack" working down and to the right from the upper-left corner of the screen.

For now, let's just specify a window's location and let the size be the application default. The x and y offsets can be either positive or negative. If you specify positive offsets, you're positioning the left side and top side of the window. Negative offsets are interpreted differently. The possible values for the x and y offsets and their effects are shown in this table:.

Table 3-1. Geometry Specification x and y Offset

Offset Variables	Description
`+xoff`	A positive x offset specifies the distance by which the left edge of the window is offset from the left side of the display.

Table 3-1. Geometry Specification x and y Offset

`+yoff`	A positive y offset specifies the distance by which the top edge of the window is offset from the top of the display.
`-xoff`	A negative x offset specifies the distance by which the right edge of the window is offset from the right side of the display.
`-yoff`	A negative y offset specifies the distance by which the bottom edge of the window is offset from the bottom of the display.

For example, the command line:

```
% xclock -geometry -10+10 &
```

places a clock of the default size in the upper-right corner of the display, 10 pixels from the top and right edges of the screen.

To place a window in any of the four corners of the screen, flush against its boundaries, use the following x and y offsets.

Table 3-2Geometry Specifications for Corners

Offset Specification	Window Location
`+0+0`	Upper-left corner of the display.
`+0-0`	Lower-left corner of the display.
`-0+0`	Upper-right corner of the display.
`-0-0`	Lower-right corner of the display.

If you want a window placed away from one or both edges of the screen, the guesswork starts. How many pixels away from the left side of the screen? How many pixels down from the top? You'll have to experiment with placing some clients on your screen to get a better idea of x and y offsets. And you'll have to rerun the experiments when you move to a different workstation with a different display board. This is why OPEN LOOK provides a stacked window placement policy, as described earlier.

It's actually a good idea to start some windows and move them around on your screen using the pointer. You can position some windows in different places by dragging them, and watching their sizes and positions. Under *olwm* you need only set the resources

```
OpenWindows.showMoveGeometry: True
OpenWindows.showResizeGeometry: True
```

Then *olwm* will display the position during a move, and the size during a resize. This information is displayed in a tiny window in the upper-left corner of the screen.

If for some reason you are not using *olwm*, you must figure out the window's x and y offsets using the *xwininfo* client. Chapter 8, *Other Standard Clients*, describes *xwininfo*, which gives you several statistics about a window including the coordinates of its upper-left corner.

Now what about the size of a window? For *xterm*, the size of the window is measured in characters and lines (by default 80 characters by 24 lines). If you want to use a large VT100 window, say 100 characters wide by 30 lines long, you could use this geometry specification:

```
% xterm -geometry 100x30-0-0 &
```

This command creates a large *xterm* window in the lower-right corner of the screen, as illustrated in Figure 3-2.

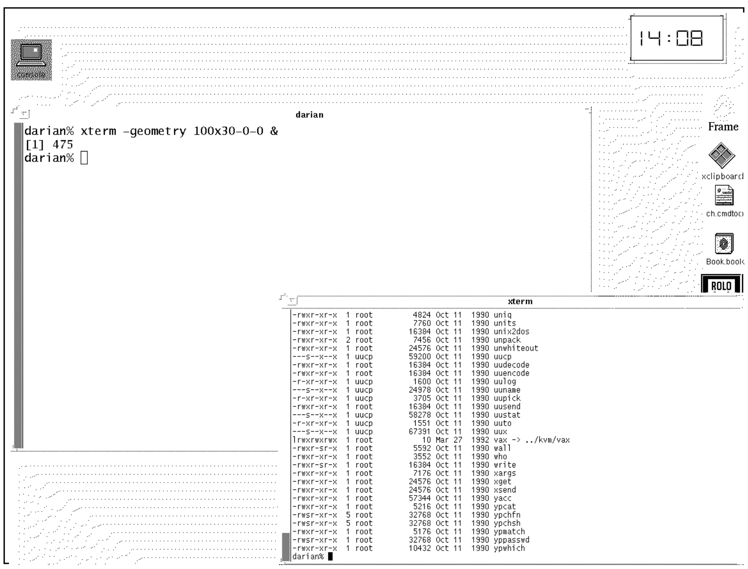

Figure 3-2. xterm window sized and positioned with the -geometry option

As stated previously, most of the standard clients (other than *xterm*) are measured in pixels. For example, *xclock* is 164 pixels square by default (exclusive of the window manager's decorations or frame). A client's default dimensions may appear on its reference page in Part Three of this guide. However, you'll probably need to experiment with specifying sizes (as well as locations) on your display. (See Chapter 8, *Other Standard Clients*, and the client reference page, for more about *xclock*.)

The geometry option is not the only means available to specify window size and location. Several clients, including *xterm*, allow you to set the size and location of a window (and often its icon or an alternate window) using resource variables (in an *.Xdefaults* or other defaults file). We'll introduce some of the basics of specifying resources later in this chapter. See Chapter 12, *Setting Resources*, for more detailed instructions on setting resources. See the appropriate client reference pages in Part Three of this guide for a complete list of resources used by any given client.

Clients built using the XView toolkit can also accept the command-line options `-Wp` x y to specify the window position in pixels, and `-WP` x y to specify its position when iconified.

✗ These XView-specific forms do not accept the + or - forms of position to indicate distance from the right margin or bottom. Furthermore, a bug in the XView library in OpenWindows 3.0 prevents use of the abbreviated "+xoff+yoff" forms of `-geometry` with negative values of *xoff* or *yoff*, that is, for setting the location relative to the bottom or the right margin. By experimentation we've determined which of the following forms work:

```
perfmeter -geometry -10-10fails (centre of screen!)
perfmeter -geometry 64x64-10-10 works (lower left corner)
perfmeter -geometry +10+10 works (upper left corner)
perfmeter -geometry +10-10 fails (10 from left, but centry of screen)
```

Accordingly, we recommend using the full-blown specification, even though it means hard-coding the size, with these clients, until the next release of the XView library.

✗ You should be aware that, as with all user preferences, you may not always get exactly what you ask for with any geometry request in any client. The X Window System requires that clients be designed to work with any window manager, not just *olwm*. If you are using some other window manager, it may have its own rules for window or icon size and placement. However, priority is always given to specific user requests, so you won't often be surprised.

3.2.2 Running a Client on Another Machine: Specifying the Display

We have yet to take advantage of X's networking capabilities. Remember that X allows you to run a client on a remote machine across a network. Generally, the results of a client program are displayed on a screen connected to the system where the client is running. However, if you are running a client on a remote system, you probably want to see the results on your own display (connected to a local server).

Running a client on a remote machine may give you access to different software, it may increase the efficiency of certain processes, or benefit you in a number of other ways. We discussed some of the advantages of running a client on a remote machine in Chapter 1, *An Introduction to OPEN LOOK and the X Window System*. See the section *X Architecture Overview* for details.

But how does running a client on a remote system affect the way you work with the X display? Once a client is running, it doesn't. You can display the application window on your own screen, enter input using your own keyboard and pointer, and read the client's output in the window on your screen—all while the actual client process occurs on another machine.

However, by default a client creates its window on a display connected to the machine on which it is running. In order to run a client remotely and display its results locally, you must tell the client process where to display its window. For this purpose, X provides the `-display` command line option. Like `-geometry`, the `-display` option is recognized by almost all X clients. The display option tells the client on which server to display results (i.e., create its window). The option has the syntax:

`-display` [***host*****:*****server***[.***screen***]]

In some clients the `-display` option can be abbreviated as `-d`, but it's safest to use the full form.†

The argument to the display option is a three-component *display name*. The ***host*** specifies the machine on which to create the window, the ***server*** specifies the server number, and the ***screen*** specifies the screen number.

In this context, "server" refers to a single physical display controlled by one X server program. (A display may be composed of multiple screens, but the screens share one keyboard and pointer.) Most workstations have only one keyboard and pointer and thus are classified as having only one display. Multi-user systems may have multiple independent displays, each running a server program. If only one display exists, as is the case with most workstations, it is numbered 0; if a machine has several displays, each is assigned a number (beginning with 0) when the X server for that display is started.

Similarly, if a single display is composed of multiple screens (sharing one keyboard and pointer), each screen is assigned a number (beginning with 0) when the server for that display is started. Multiple screen displays may be composed of two or more physical monitors. Alternatively, two screens might be defined as different ways of using the same physical monitor. One example is the Sun-3/60, Sun-3/110 and Sun-4/110 color workstations, described in Section 2.1.2, "Logging In at a Full Screen Prompt: Starting OpenWindows or X." PC/386 systems often support "virtual terminals;" on these you can sometimes run a separate X server on each of several virtual terminals. It is not clear whether there is a standard yet for referring to these as separate displays or separate screens; one expects that since there is but one keyboard, the server would treat it as one display with multiple screens.

Note that the ***server*** parameter of the display option always begins with a colon (a double colon after a DECnet node)‡, and that the ***screen*** parameter always begins with a period. If the host is omitted or is specified as `unix`, the local machine is assumed. If the screen is omitted, screen 0 is assumed.

Although much of the current X Window System documentation suggests that any of the parameters to the `-display` option can be omitted and will default to the local node, server and screen 0, respectively, we have not found this to be true. In our experience, only the ***host*** and ***screen*** parameters (and the period preceding ***screen***) can be omitted. The colon and ***server*** are necessary in all circumstances. For local hosts, the minimum form of the `-display` argument is thus *:0* and this form on some implementations may be more efficient than using *unix:0*, which some other documentation advises.

In summary, the display name can have the following forms:

† "X Toolkit" clients can usually use -d, but XView clients cannot because of the options '-disable_retained' and '-disable_xio_error_handler'. See the Reference Manual page for 'XView' in Part Three of this Guide.

‡ By convention, DECnet node names end with a colon.

On UNIX systems, the display name can also be stored in the DISPLAY environment vari-

Table 3-3. Display Syntax Forms

Display	Where window appears	How connected
(none)	Console of workstation you're using	Local socket (*/tmp/.X11-unix*)
:0 or :0.0	As above	As above
:0.1	Screen 1 of console	As above
unix:0.0	Console	As above
hostname:0.0	Console or screen of hostname	TCP/IP network (usually Ethernet)
hostname:0	As above	DECnet

able. Clients running on the local system access this variable to determine which display to connect to. The DISPLAY variable is set automatically, both by the OpenWindows start-up script and by the *xterm* terminal emulator. OpenWindows defaults it to `:0`, meaning the local host. In most circumstances, *xterm* sets the server and screen numbers to 0, and either omits a hostname (the local host is assumed) or sets the hostname to "unix," a generic name, which also defaults to the local host.

If you only run processes on the local machine, you don't have to deal with issues concerning the display setting. Clients running locally access the DISPLAY variable and open windows on a display connected to the local host. However, if you want to run a process on a remote machine and display the results locally, things become more complicated. A client running on a remote machine does not have access to the DISPLAY variable on the local machine. By default, a client running on a remote machine checks the DISPLAY setting on *that* machine, where DISPLAY may not even be set!

You can override the DISPLAY environment variable a client accesses by using the `-display` option when you run the command. Think of `-display` as a pointer to the physical display on which you want the window to appear. For example, say you're using a single display workstation and the display also has only one screen. The hostname of the workstation is *kansas*. In order to tell a client to connect to a display, you must identify it by its unique name on the network. (You cannot identify your display by the common shorthand—`unix:0.0`, `:0.0`, or some variation.) Let's assume that the complete display name for the workstation *kansas* is:

```
kansas:0.0
```

Now let's say you want to run an *xterm* window on a faster system—let's call it *oz* —on your network. In order to run an *xterm* on *oz* but display the window on your screen connected to *kansas* (the local server), you would run the *xterm* command using a remote shell (*rsh*):†

```
% rsh -n oz xterm -display kansas:0.0 &
```

The *xterm* process runs on *oz*, but you've directed the client to use the display and screen numbered 0 on *kansas*, your local system. Notice that `kansas:0.0` is the complete display name. If the workstation (*kansas*) has only one screen or it has multiple screens but you want to specify screen 0, you can omit the screen number and the preceding period (`.0`).

Keep in mind that for this process to succeed, you must have permission to run commands on the remote system (*oz*) *and* the remote system must have permission to "open" the local display (on *kansas*). If you don't have permission to run commands "over there," you will get back the message

```
Permission denied.
```

or words to that effect.

Normally you have permission to create windows from any machine on your local network onto your display, assuming that your local network shares home directories. The file *.Xauthority* in your home directory is created automatically by the *openwin* script and any other modern implementation of X11; this file grants permission to access your screen. It is not publicly readable so that other users cannot copy it. If your home directory is available on the remote machines from which you wish to launch clients (for example, a home directory mounted via NFS to a group of machines), no further action is necessary. But if you wish to grant permission to another account that has a *different* home directory but that you are "host equivalent" to, you can do so with the *rcp* command:

```
rcp .Xauthority otherhost:
```

If your account on *oz* has not been granted access to the server running on *kansas*, the window will not be opened, and you should get an error message similar to:

```
Xlib: connection to "oz:0.0" refused by server
Xlib: Client is not authorized to connect to server
Error: Can't Open display
```

On OpenWindows, this *additional* message will appear on the console of *oz*:

```
X11/NeWS Network security violation
 Rejected connection from: kansas (192.31.6.192)
For more information, see the xhost(1) and xauth(1) man pages
```

† The command to run the remote process might be different depending on the available networking software. Some versions of UNIX include a so-called "restricted shell" called *rsh*, so these systems have to invent a new name for the real *rsh*. If it's not there, or if you get the message `rsh: cannot open oz`, try *rcmd* or *remsh*. Ask your system administrator for the proper command.

If your command fails with this message, and you can't provide a *.Xauthority* file, then as a last restor try entering the command:

```
% xhost +oz
```

in a terminal emulator window running on your display. Then run the remote shell (*rsh*) again.

✗ Be aware that now *anybody* with an account on `oz` can create windows on your display; this is why the *.Xauthority* mechanism mentioned above is preferable. If problems persist, consult the *xhost* and *xauth* reference pages, or your system administrator, or refer to Volume Eight, *X Window System Administrator's Guide*.

This problem will also occur if you *su* to *root* and then try to create windows. Some vendors' software installation documentation tells you to use the "xhost +" form, but this is insecure. A better method is:

1. su to root
2. Copy *.Xauthority* from your home directory to */*.
3. Start the installation window.
4. Remove the file created in step 2!
5. Exit the root shell
6. Do the installation in the installation window.

The following command illustrates another common mistake:

```
% rsh oz cmdtool &
```

If you try to run a client using a remote shell and forget to direct the client to create its window on your own display, the window will not be displayed and you'll get an error message stating that the display cannot be opened. Worse, the terminal *may* appear on oz's console if an X server is running there; this gives whoever is logged in there access to your files on that machine!

In addition to specifying a local display, if permissions allow, you can also use the display option to open a window on someone else's display. You might want to display a window on another user's screen for instructional purposes. Multi-user systems can even be set up to allow teachers to display educational material simultaneously to several students, each using an X display of some sort.

If you're working on *kansas* and you want to open a terminal window on the first display connected to *oz*, you could use the command:

```
% cmdtool -display oz:0.0 &
```

Note that you can only open a window on another display if the server running that display permits your client access. (Access must be granted from the remote server, perhaps using *xhost*.) If *oz* does not allow *kansas* access, this command will fail and an error message will indicate that the display cannot be opened.

3.2.3 Once You Run a Remote xterm using -display

A less than obvious repercussion of using `-display` to run a remote *xterm* is that the option sets the DISPLAY variable for the new *xterm* window—and that DISPLAY setting is passed on to all child processes of the client. Therefore, once you run an *xterm* on a remote system and correctly specify your own display, you can run any number of clients from that *xterm* and they will all be displayed on your screen automatically (no `-display` option is necessary).

✗ This automatic propagation is not done by the current version of *cmdtool* or *shelltool*

In one of the examples in the preceding section, we ran an *xterm* on the remote system *oz*, specifying the local display `kansas:0.0` with the `-display` option. To query the contents of the DISPLAY variable in the resulting *xterm*, use the command:

```
% echo $DISPLAY
```

The system should echo:

```
kansas:0.0
```

verifying that the display name has been passed to the DISPLAY variable in the new *xterm* window. You can then run any client you want on *oz* by entering the command in this *xterm* window and the window will automatically be displayed on `kansas:0.0`. The DISPLAY setting will also be passed to any children of *this* process as well, and will be propagated for any number of "generations."

3.2.4 Logging In to a Remote System

If you log in to a remote UNIX system using *rlogin* (or *telnet*) in a terminal emulator window, it's a good idea to set the DISPLAY variable in the new shell to reflect your local display. Then if you run a client process from this window, the new window will be placed on your local display and the DISPLAY setting will be passed on to all child processes.

When you set the DISPLAY variable from the command line, the syntax varies depending on the UNIX shell running. The following command sets the variable under the C shell.

```
% setenv DISPLAY kansas:0.0
```

To set the DISPLAY variable under the Korn or Bourne shell, use:

```
$ DISPLAY=kansas:0.0; export DISPLAY
```

3.2.5 Complications: LD_LIBRARY_PATH

Whenever you are starting a client program, you need to be sure that the shell variable LD_LIBRARY_PATH has been set. $LD_LIBRARY_PATH is the SunOS shared library search path, and is needed by most of the OpenWindows clients. This is normally set for you by the OpenWindows script, and normally propagated for you by the *rlogin* program.

Occasionally this fails, and programs refuse to start up.When this is the problem, you will get the message like

```
ld.so.1: programname: can't find file libxview.so.3
Killed.
```

Here are some circumstances in which you must set it manually

- You aren't using the *openwin* script to start your environment
- You have logged in remotely using *telnet* or any other network program that doesn't propagate your "environment variables"
- You are logging in from a non-Sun platform to a Sun platform to run some OpenWindows clients.

In these cases, the command

```
% setenv LD_LIBRARY_PATH /usr/openwin/lib
```

or

```
$ LD_LIBRARY_PATH=/usr/openwin/lib; export LD_LIBRARY_PATH
```

(typically from within a shell startup script) will solve the problem.

3.2.6 Monitoring the Load on a Remote System

There are two clients used to track the system load average on remote machines. These clients will be useful in determining which remote machine(s) on your network are least heavily loaded and therefore most suitable for logging into to get work done. The MIT client *xload*, described in Chapter 8, *Other Standard Clients*, tracks load average from one machine; the OpenWindows DeskSet client *perfmeter* described in Chapter 7, *The OpenWindows DeskSet Clients*, tracks several load-related statistics from a system.

3.2.7 Other Command Line Options

In Section 3.2.1, "Window Geometry: Specifying Size and Location" and Section 3.2.2, "Running a Client on Another Machine: Specifying the Display", we saw how to use the `-geometry` and `-display` options, which are accepted by most clients. Chapter 9, *Command Line Options*, describes some of the other options accepted by most of the standard clients. These options set window features such as:

- The font in which text is displayed.
- The background color.
- The foreground color (such as the color of text).
- The text displayed in the title area.
- The text of an icon label.

Many clients also accept a large number of toolkit-specific options (listed in Chapter 11, *Command-line Options*) and application-specific options (listed on the reference page for each client in Part Three of this guide). Using a combination of standard and application-specific options, you can tailor a client to look and behave in ways that better suit your needs.

Command line options override the default characteristics of a client for the *single* client process. Traditional UNIX applications rely on command line options to allow users to customize the way they work. X also offers many command line options, but these options have some limitations and liabilities.

First, the number of client features that can be controlled by command line options is limited. Most applications have many more settable features than their command line options indicate. Actually, a client can have so many features that typing a command line to set them all would be impractical. And if you generally use the same options with a client, it is tedious (and a waste of time) to type the options each time you run the program.

X offers an alternative to customizing a single client process on the command line. You can specify default characteristics for a client using variables called *resources*, described in "Customizing the X Environment: Specifying Resources" on page 69.

Finally, remember that almost all clients will accept a `-help` option to give at least a summary of their command line options.

3.3 Putting it All Together

Now that we've learned something about the tools of the display, how to size and position windows, and run remote processes, let's try to set up a useful working display.

Say we're using a workstation with the hostname *jersey*. The workstation has a single display with one screen, so it will be called screen 0. Once X and the window manager are running, we might set up the display using the following commands.

First, run an *xterm* on a more powerful remote system called *manhattan* and place it on screen 0 of *jersey*.

```
% rsh manhattan xterm -geometry +0-0 -display jersey:0 &
```

Run *perfmeter* windows on both *jersey* and *manhattan* to monitor loads on these systems. Again, place the windows on *jersey*'s color screen in convenient locations.

```
% perfmeter -geometry 64x64-10-200 &
% perfmeter manhattan -geometry 64x64-10-20 \
        -display jersey:0 &
```

Run another *xterm* window on *jersey*.

```
% xterm -geometry +50-0 &
```

Then iconify the login *xterm* window so that you don't inadvertently kill it (and shut down X in the bargain). Remember: to iconify a window place the pointer on the Window Mark on the window's frame and click the SELECT button. (See "Moving, Resizing and Iconifying Windows" earlier in this chapter.)

Run an *oclock*.

```
% oclock &
```

Figure 3-3 shows the *jersey* display, a fairly useful layout.

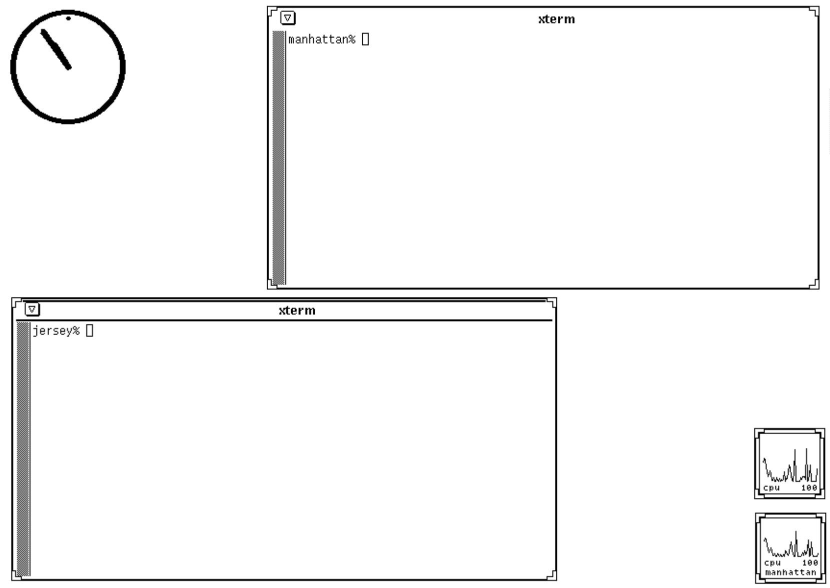

Figure 3-3. A working display

As we'll see in "Customizing your Session Start-up" on page 72, on most systems you can place the commands you run to set up your display in a special file that is invoked when you log in. Once this file (usually called *.xinitrc*, *.openwin-init*, or *.xsession*) is in place, your display will be set up to your specifications automatically each time you log in.

Notice that the commands we used to set up *jersey* illustrate the power of the `-geometry` and `-display` options to create a working environment that suits individual needs. However, these options barely hint at the number of features you can specify for each client. The following section introduces some principles of client customization. Part Two of this guide examines customization in depth.

3.4 Customizing the X Environment: Specifying Resources

As we've seen, you can use command line options to specify certain characteristics for a single client process. In a sense, command line options allow you to customize one program.

X11 also provides a mechanism that allows you to specify characteristics that take effect *every* time you run a client. Almost every feature of a client program can be controlled 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.

You generally place resource specifications in a file in your home directory. (The file can have any name, but is often called *.Xresources* or *.Xdefaults*.) The resources you specify are one of several factors that affect the appearance and behavior of a client.

☞ A useful tool that the OPEN LOOK GUI provides is a *properties* program to edit your resource files for you, that is, to customize many aspects of your display. *Properties* is very easy to use, much easier than editing the resource files. The *properties* program is described in Section 14.1, "Properties Resource Editor" on Page 331. Accordingly, *users of the OPEN LOOK GUI on Sun OpenWindows do not need to learn about resource editing to configure the standard OPEN LOOK clients.* However you will eventually need to learn about resource editing for modifying extended features of the standard clients as well as most non-OPEN LOOK clients.

By default, the way a client looks and behaves is determined by the program code, and in some cases, by a system-wide file of *application defaults*. Several clients have application defaults files that determine certain client features.†

Application defaults files generally reside in the directory */usr/lib/X11/app-defaults* (on OpenWindows, in $OPENWINHOME*/lib/app-defaults*) and have the same name as the client application (except that the first letter is capitalized, and if the first letter is an X, the second letter is also capitalized). You can also have an application defaults file in your home directory, with the same format name. For example, the files */usr/openwin/lib/app-defaults/XEdit* and *$HOME/Bitmap* would both be valid names for application defaults files.

Need brief discussion of XFILESEARCHPATH—here, or in Resources chapter?

In describing the appearance and behavior of clients in this guide, we assume all of the standard application defaults files are present on your system and accessible by the client programs. If, by some chance, a client's application defaults file has been edited or removed from your system, the client may not look or behave exactly as we describe it. If a client application appears substantially different than depicted in this guide, you may be using a different version of the program or the application defaults may be different. Consult your system administrator if this causes problems.

Within an application defaults file, defaults are set using resources. The resources specified in a client's application defaults files are usually just a subset of a greater number of resources that can be set.

If the characteristics you set in your own resources file already have system-wide application defaults, your own settings take precedence. Keep in mind, however, that command

† For *xterm*, as for many clients, the application defaults let you control such things as the labels for menu items, the fonts used to display menu items, and the shape of the pointer when it's in an *xterm* window.

line options override both your own defaults and any system-wide defaults for the single client process.

To make your resource specifications available to all clients, X provides a program called *xrdb*, the X resource database manager. This program stores resources directly in the server where they are accessible to all clients, regardless of the machine the clients are running on. If you do not use *xrdb*, then each program reads the file *.Xdefaults* and (if it exists) *.Xdefaults-**host*** (where ***host*** is the name of the workstation whose display is being used), from your home directory. This is done by the lower-level "Xlib" routines, so it should work with all toolkits.

In some cases, a resource variable controls the same characteristic as a command line option. However, while the option specifies a characteristic for the single client process being invoked, a resource variable makes the characteristic the program default.

Each client recognizes certain resource variables that can be assigned a value. The variables for each client are listed on its reference page in Part Three of this guide.

A resource definition file is basically a sequence of lines—you can think of it as a two-column list—where each line specifies a different resource. The simplest resource definition line has the name of the client, followed by an asterisk, and the name of the variable, followed by a colon, in the left column. The right column (separated from the left by a tab or whitespace) contains the value of the resource variable.

```
client*variable: value
```

The following example shows five simple resource specifications for the *xclock* client. These particular resources specify attractive colors for *xclock*. See *Chapter 11, Command-line Options,* and the *xclock* reference page in Part Three for details.

Example 3-1. Resources to create a custom xclock

```
xclock*hands:              green
xclock*highlight:          royalblue
xclock*background:         lightblue
xclock*update:             1
xclock*chime:              true
```

To set up your environment so that these characteristics apply each time you run *xclock*, do the following:

1. In your home directory, create a file containing the resources listed in Example 3-1. Name the file *.Xdefaults*. A resource file can actually have any name, but is often called *.Xresources* or *.Xdefaults*. *.Xdefaults* is preferable under OpenWindows since this file is maintained by the *Properties* program.

2. Load the resources into the server by entering the following command in a terminal window:

```
% xrdb -load .Xdefaults
```

Then each time you run *xclock* without options (for the remainder of that login session), the window will reflect the new defaults.

You should load resources using *xrdb* every time you log in. In the next section, we'll describe how to automate this process using a special start-up script, which also opens the client windows you want on your display.

If you want to run an application with different characteristics (colors, update frequency, etc.) than the defaults, use the appropriate command line options to override the resource specifications.

Resource specifications can be much more complicated than our samples suggest. For applications written with an Intrinsics-based toolkit (such as the MIT Athena Widgets, AT&T's Open Look Intrinsics Toolkit, or the Open Software Foundation's Motif Toolkit), X allows you to specify different characteristics for individual components, or *widgets*, within the application. Typical widgets create graphical features such as menus, command buttons, dialog boxes, and scrollbars. Within most toolkit applications is a fairly complex widget hierarchy—widgets exist within widgets (e.g., a command button within a dialog box).

Resource naming syntax can parallel the widget hierarchy within an application. For instance, you might set different background colors for different command buttons and specify still another background color for the dialog box that encloses them. In such cases, the actual widget names are used within the resource specification. Chapter 12, *Setting Resources*, explains the resource naming syntax in greater detail and outlines the rules governing the precedence of resources.

3.5 Customizing your Session Start-up

By now you know that there has to be a file specifying the clients you want to run each time you login, otherwise you would have to start all your clients by hand. If you just start the X server itself, it will not start *any* clients for you, and you'll have to start the first terminal window from someplace else—not very convenient! The program *xinit* was designed to help you start all the clients you need. If you start *xinit* with no special preparation, it will try to start the default X server and at least one terminal emulator, but no window manager. Normally you would have a file called *.xinitrc* to tell *xinit* what clients to run. After *xinit* had been in use for a while, another program, *xdm*, was intended to solve the same problem in a different way. The *xdm* program is more sophisticated, but more complicated. Normal users need only be aware of one file that it uses, though: the *.xsession* file. Like the *.xinitrc* file, this file is a list of the clients that you want run each time you login via *xdm*.

Let's look at these files, and some others that are important for customizing your window system environment. Bear in mind that we are just going to scratch the surface here. For the full scoop on *xinit, xdm* and all their customization, you should refer to Volume Eight, *X Window System Administrator's Guide*.

The *.xinitrc* and *.xsession* files contain the clients you want run. But this script—whichever of the two you are using—should also:

- Set the DISPLAY environment variable
- Load your X resources file with *xrdb*.
- Start the window manager *olwm*.
- Start any terminal emulator windows, desk accessories, DeskSet tools, or X applications (text processors, spreadsheets, . . .) that you want to run.

Most of your clients will not interact directly with *xinit* or *xdm*, so they should be run "in the background", that is, with an ampersand ("&") at the end of each command. If you forget this, the shell (command interpreter) that is processing your start-up file will wait until the non-backgrounded client waits, before starting the remaining clients. However, if you background *everything* in the script, the shell will get to the end of the file and, with nothing more to do, terminate itself or "exit". This will cause the X server to terminate. Because of this, you may need to run a single console terminal window that is not put into the background. Alternately, you can use the *wait* command as the last line of your script.

Example 3-2 contains a listing of a simple example *.xinitrc* file:

Example 3-2. Simple .xinitrc file

```
#!/bin/sh

DISPLAY=myworkstation:0; export DISPLAY

# xterm -geometry 80x6+0+75 -T Console -C &
contool -Ws 550 160 -Wp 0 0 -WP 5 80 -Wi &

xrdb -load < $HOME/.Xdefaults
xset m 8 fp+ /usr/local/lib/fonts # bc

olwm &

clock -Wn -Wp 930 90 -digital -24 &

# Some terminal windows for working in.
cmdtool -Ww 80 -Wh 24 -Wp 000 200 -Wl "$hostname" -scale extra_large &
#xterm -geometry 80x48+000+176 -T "$hostname"&
#xterm -geometry 80x48+580+180 -T sq -e rlogin sq &

maker -geometry -0+900 & # Run FrameMaker if licensed

wait # forever, or until killall or shutdown.
```

We've used the Bourne shell (*/bin/sh*) as it is the shell most commonly used for writing scripts. You can use another shell such as *csh* if you are more comfortable writing scripts in it.

It is common to find commented-out entries (lines that begin with a "#") when looking at another person's *.xinitrc* file, so we've left some in. Such lines are ignored by the system.

The main thing is that all the possible command-line options discussed previously (as well as those in Chapter 11, *Command-line Options*) can be used, so this mechanism represents a powerful and flexible facility. If you may be using both *xdm* and either *xinit* or the *openwin* start-up script, you should consider keeping your start-up clients in a third file that is included in either your *.xsession* or your *.xinitrc* file. This is in fact what the default *openwin* start-up files do.

As well, it is possible without using *xdm* to have your X11 session started automatically when you login, simply by invoking it from your *.login* or *.profile* startup file. Example 3-3 is a fragment of the default *.login* file used in the current releases of SunOS. This code will normally start the *openwin* script, but gives you a chance to interrupt it first (during the `sleep 5` sequence), and logs you out afterwards, again with a chance to interrupt. This has proven to be quite convenient.

Example 3-3. SunOS .login code to start OpenWindows at each login

```
switch( $mychoice ) # "mychoice" is either "openwin", "sunview" or null
case openwin:
    echo -n "Starting OpenWindows (type Control-C to interrupt)"
    sleep 5
    $OPENWINHOME/bin/openwin    # run the window system
    clear   # get rid of annoying cursor rectangle
    echo -n "Automatically logging out (type Control-C to interrupt)"
    sleep 5
    logout  # logout after leaving windows system
    breaksw
    #
# other cases here
endsw
```

You may wish to incorporate code similar to this in your shell-specific startup script.

3.5.1 OpenWindows Specifics

OpenWindows uses a variation on the above files. If you already have a *.xinitrc* file, you can continue to use it, since OpenWindows does use *xinit* to start its server. But you should add a few entries gleaned from reading the OpenWindows-provided files mentioned below, otherwise certain aspects of OpenWindows's interface may not behave correctly.

If you do not have a *.xinitrc* file in your home directory, then OpenWindows will create one for you. The file it creates is fairly short; it calls an extra file named *.openwin-init* to start your clients. The file it creates is copied from */usr/openwin/lib/Xinitrc*, and does the following steps:

- Runs *xrdb* on *$HOME/.Xdefaults* or *$OPENWINHOME/lib/Xdefaults*
- Includes *$OPENWINHOME/lib/openwin-sys*, another script
- If you haven't requested otherwise, enables SunView binary compatibility mode.
- Starts the OPEN LOOK Window Manager and waits until it has finished initializing

- Runs the client start-up script, either *$HOME/.openwin-init* or *$OPENWIN-HOME/lib/openwin-init.*
- Waits for *olwm* (the "key client" or "session gate client") to exit.

While this may seem like substantial added complexity, it has two advantages:

- It works correctly even if the user has no start-up files to begin with, and
- It is easy to update the list of clients and their arguments, since they're in a separate file.

This technique of having the clients separate from the setup information is a generally useful one. It can also be used with any of OpenWindows, *xinit*, or *xdm*.

In fact, this is the reason that the OPEN LOOK Window Manager has a built-in operator called "SAVE_WORKSPACE", called from its default `Utilities` menu, that saves the list of all known X clients, their positions, and their arguments, into your *.openwin-init* file. OpenWindows users can therefore forget all about *.xinitrc* files, and simply use the "SAVE_WORKSPACE" mechanism to keep their list of clients. Or, they can use all the flexibility that the *.xinitrc* file provides, by using their own file in place of the one that OpenWindows provides for you.

✗ The "SAVE WORKSPACE" option produces a file that may not contain certain non-OPEN LOOK, non-X11 clients, and that may contain additional detail beyond what you would normally want to specify.

You should also know about the OPEN LOOK Window Manager menu file, *.openwin-menu,* used to control the workspace or root menu from *olwm*. This is described in detail in Chapter 13, *Customizing olwm*

3.6 Where to Go From Here

We've introduced some basic operations you can perform using OPEN LOOK and the *olwm* window manager. There are many useful client programs supplied with OPEN LOOK and the X Window System. Details of how to use two of the most important clients, file manager and the terminal emulator, are provided in Chapter 4, *Using the OPEN LOOK File Manager*, and Chapter 5, *The cmdtool/shelltool Terminal Emulator* (the MIT terminal emulator is described in Appendix A, *The xterm/olterm Terminal Emulator*). For instructions on performing additional window manager operations, read Chapter 6, *Using the OPEN LOOK Window Manager*. An overview and tutorial for OpenWindows and other standard clients is provided in Chapter 7, *The OpenWindows DeskSet Clients*, and Chapter 8, *Other Standard Clients*. Chapter 9, *Graphics Clients*, describes several graphics utilities available with X. Part Two of this guide provides information on customizing your X environment. All clients are described in detail in a reference page format (basically the UNIX *man* page format) in Part Three of this guide.