CHAPTER 14

Customization Clients

This chapter describes how to personalize the appearance of your display, and the operation of your keyboard and pointer, using these clients:

Properties The OPEN LOOK Properties Editor (Workspace Manager) to update your X Resources file for such things as color settings, icon placement, menu operations, mouse settings and other features.

xset A standard X11 client to set certain characteristics of the keyboard, pointer, and display.

xsetroot A standard X11 client to set root window characteristics.

xmodmap A standard X11 client to change pointer and modifier key mappings.

xkeycapsA client to help with key mappings.

14.1 Properties Resource Editor

The OPEN LOOK properties editor is both a good example of an OPEN LOOK application and a useful tool that edits your X resources file under control of OPEN LOOK-based menus. It differs from the Motif Resource Editor *mre* described in Appendix M, *OPEN LOOK and Motif*, in that it is a fully supported part of the OPEN LOOK package, and in that it incorporates color resource editing similar to that of the contributed-software *xcol* program. There are several main categories of properties that you can edit:

Table 14-1. Property Categories

Category Name	Function	Restriction
Programs	Root Menu User Programs	AT&T-OL only
Color	Window, root window, data areacolors	Color monitor only

Table 14-1. Property Categories

Category Name	Function	Restriction
Icons	Icon placement	
Menus	Menu operations	OpenWindows only
Mouse Setting	Mouse speed, pointer jumping, etc.	
Mouse Modifiers	Mouse to OPEN LOOK mappings	AT&T-OL only
Keyboard Functions	Keyboard shortcuts	AT&T-OL only
Miscellaneous	Pointer focus, scrollbar side, etc.	

We'll discuss each of these in order, after showing you how to start up the properties editor.

14.1.1 Starting The Properties Editor

The Properties Editor, called the Workspace Manager Application in the AT&T documentation, can be started from the *Properties* entry in the *Workspace Menu* or root menu on either AT&T-OL or Sun OpenWindows.† The Sun OpenWindows Properties Editor can also be started from a shell window under the name *props*, i.e.,

```
props &
```

You should get an initial screen like Figure 14-1 , although the initial category will vary

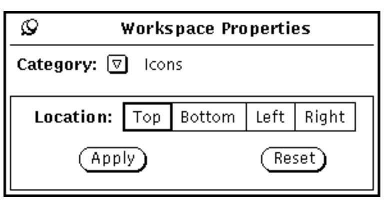

Figure 14-1. Properties Editor initial screen

from one release to the next. From here you can select any one of the categories by pulling down the abbreviated menu button, like so:

Note that on each category panel there is an *Apply* button. When you press this, two things happen. One is that your *.Xdefaults* file is rewritten; the other is that the new resource values are written as an X11 "property" on the root window, so that both *olwm* and any OPEN LOOK applications will notice them at once.

† On OpenWindows, it may not be available from the root menu if you have modified the menu file as described in Chapter 13, *Customizing olwm*.

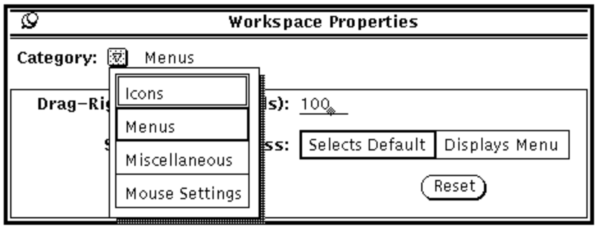

Figure 14-2. Abbreviated Menus on Properties.

The rewriting of the resource file *.Xdefaults* is not perfect. For one thing, the file is processed with the C language pre-processor as part of this re-writing, and a side effect of this is that any comments you have in it will be lost. Older OpenWindows versions thereforegive you a Notice with this text:

```
Applying your changes will modify your ~/.Xdefaults file.
All comments in the file will be lost.
Do you want to do this?
```

The buttons on the Notice are *Yes* and *No*. If you click SELECT on *Yes*, your file will be rewritten, and (true to its word) all comments will be lost†. And the new properties will be applied to windows on the display. On the other hand, if you select *No*, no changes will be made, either to your file or to the root window property. If you don't do a lot of work on your *.Xdefaults* file, you may not care about comments in it.

Newer OpenWindowsversions of the Properties Editor instead update the *.OWdefaults* file in your home directory, which is normally loaded with *xrdb* as described in Chapter 12, *Setting Resources*.

Each panel also has a *Reset* button; note that this resets to your current value, not to the "factory default". The AT&T-OL version also has *Reset to Factory* on its categories, which resets the values in that category to the OPEN LOOK defaults.

You can select any of the categories for editing at any time. You don't have to *Apply* the changes from each panel; you only need to click the *Apply* button once before leaving the editor to save all your changes.

Notice also that the OpenWindows version comes up as a pinned, pop-up window, so it has the pop-up *Window* menu. To quit this program, you can either select *Dismiss* from the *Window* menu, or explicitly unpin-pin it by clicking SELECT on the pin to "pull the pin."

† You can fool this process by making your comments appear to be resources, as in:

```
comment.phony:this is a comment that will be preserved
```

14.1.2 Programs Menu Category

This category produces a scrolling list with the *Programs* category from the root menu. Note that if you have your own *.openwin-menu* file, it must include a file named *.openwin-menu-programs* from your home directory, or this category will be inoperative. For each item you can specify the menu label (such as *XTerm* and the command to be run (such as *xterm -fn fixed -bg pink*).

14.1.3 Color Property Category - OpenWindows

The next category is the Color Preferences menu. The OpenWindows version of the color window only appears if you are on a color display. Its appearance varies from one release of OpenWindows to another; an earlier version is shown in Figure 14-3 and the Version 3.3 window is shown in Figure 14-4 .

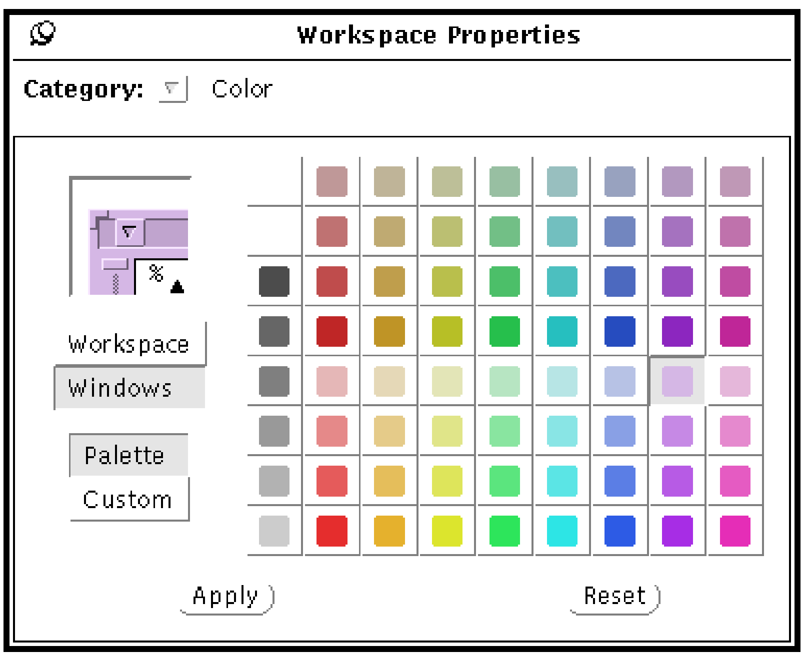

Figure 14-3. Color category - OpenWindows 3.1

The box at the center bottom (earlier versions had it in the top-left corner) of the Color category panel is a "model window" that shows a simulation of the top-left corner of a typical shell window under OPEN LOOK with the currently-selected colors. The exclusive-choice box underneath it lets you select between the OPEN LOOK Workspace (root window) color and the OPEN LOOK Window color. The "Window" color is used on window titlebars and on the background of controls such as scrollbars. The actual color of the text part of the window is not controlled by OPEN LOOK, but by the application, be it *cmdtool* or *xterm* or

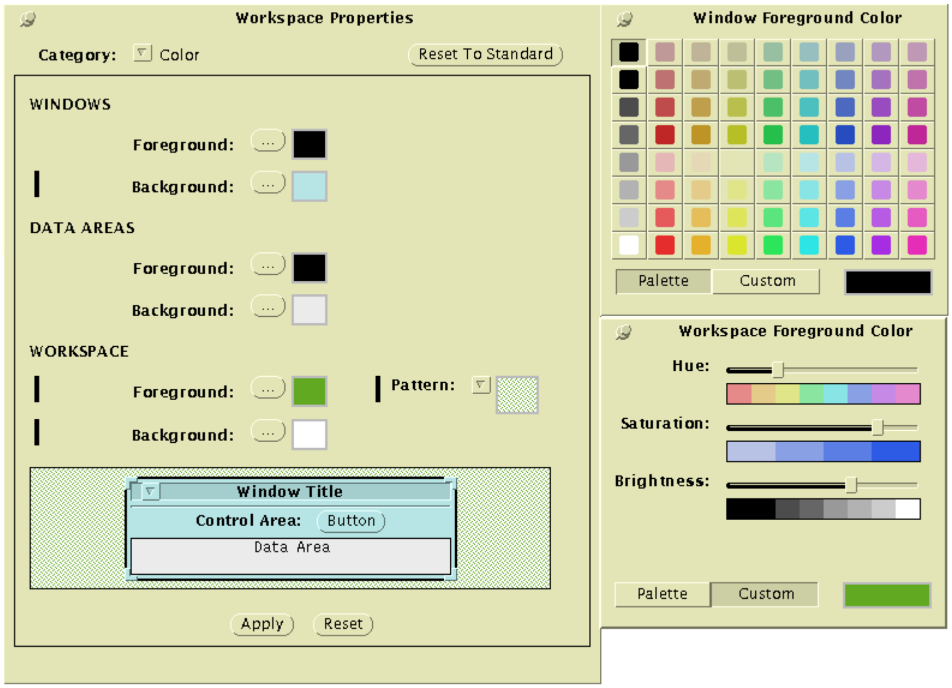

Figure 14-4. Color Category - OpenWindows 3.3

any other client. Having selected either *Workspace* or *Window* (or *Data Areas* in 3.3 or later), you can choose colors for it from among the dozens of colors displayed on the palette. Each time you pick a color, it is instantly shown on the appropriate parts of the "model window" in the upper left. Thus, you can see right away what your windows will look like if you select the *Apply* button.

You can probably find a color that you like from among the hundreds shown in the palette. But if not, don't despair. You can play artist and mix your own colors. Just select the *Custom* button, and the Palette will be replaced by a "mixer" window with sliders for Color, Saturation, and Brightness. It's difficult to describe these in a black-and-white book, so your best bet is to look at Figure 14-4 and experiment with these yourself.

Once you have the colors you like, click *Apply* to both update your *.Xresources* file and update the colors on your screen.

14.1.4 Icons Property Category

The *Icons* category lets you choose the location of icons, and in some cases the border display of icons. The screen looks like Figure 14-5.

Selecting one of the four locations will make that be the location for *olwm* to place icons. For example, *Top* means that icons will be placed in a row along the top edge of the screen.

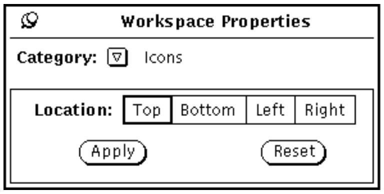

Figure 14-5. Icon Property Display.

The *Border* setting, which only shows up in the AT&T-OL version, specifies whether a border is displayed around each icon when you are on a color monitor. Under OpenWindows, you cannot control this setting. On a monochrome monitor, borders are always displayed.

As usual, click *Apply* to make your changes take effect.

14.1.5 Menus Property Category

This category only appears on the OpenWindows version. It contains only the items *Drag-Right distance* (which appears under *Mouse Settings* in the AT&T-OL version, and *SELECT Mouse Press* which appears under *Miscellaneous* category in AT&T-OL. To avoid duplication, the items are documented under the corresponding AT&T-OL entries.

14.1.6 Mouse Setting Property Category

This category has four items:

- *Multi-Click Timeout* specifies how close together two clicks of the same button can be and still be considered part of a single multi-click sequence instead of two separate clicks.
- *Mouse Damping* lets you filter out small mouse movements that may be accidental, to separate them from intentional motions of the mouse.
- *Drag-Right Distance* lets you specify how far you must drag the mouse right on a control with a menu mark in order to activate the submenu.
- *Menu Mark Region* Is the opposite; it specifies how far to the left you must move to get back to the menu mark, i.e., to pop down the submenu.

As usual, click *Apply* to make your changes take effect.

14.1.7 Miscellaneous Property Category

This is a kind of catch-all category, as you might expect by the name. It is a list of exclusive choices. The following lists the choices, their possible values, and which are currently available in the two OPEN LOOK implementations:

Table 14-2. Miscellaneous Properties

Name	Choices	Version
Beep	Always| Notices Only| Never	Both

Table 14-2. Miscellaneous Properties

Name	Choices	Version
Window layering	Individually| As A Group	AT&T-OL
Start OPEN LOOK at login	Yes| No	AT&T-OL
SELECT Mouse Press	Displays Default| Displays Menu	Note 1
Help Model	Input Focus| Pointer	AT&T-OL
Set Input Area	Click Select| Move pointer	Both
Interface Appearance	2D| 3D	AT&T-OL
Mnemonics	Off| Underline| Highlight| On-Don't Show	AT&T-OL
Accelerators	Off| On-Show| On-Don't Show	AT&T-OL
Scrollbar Placement	Left| Right	OpenWindows

Note 1: This item is in the *Miscellaneous* category in AT&T-OL, but is in the *Menus* category in OpenWindows.

14.1.7.1 Beep

Controls when the bell will ring: *Always* means whenever notices or important footer notices appear; *Notices Only* means only when a Notice appears, and *Never* means what it says.

14.1.7.2 Window layering Individually| As A Group

Controls how windows are moved to the front or back. *As A Group* means that all pop-ups move when the main window moves, while *Individually* treats each window as an independent item.

14.1.7.3 Start OPEN LOOK at login Yes|No

Controls whether the *olinit* script should be run from your *.profile* at login time.

14.1.7.4 SELECT Mouse Press Displays Default|Displays Menu

Controls what happens when you press SELECT on a control that has a menu mark. *Displays Default* means it activates the Default item for the menu. *Displays Menu* means that it pulls down or pops up the menu, but waits for you to select an item. *Displays Default* is the advanced user's choice, as it lets you get either the default (by clicking SELECT) or the menu (by clicking the MENU pointer button).

14.1.7.5 Help Model Input Focus|Pointer (AT&T-OL)

For programs written with the OLIT toolkit, controls whether HELP is offered for the text field that has the input focus, or the item that the pointer is over, when the HELP key is pressed.

14.1.7.6 Set Input Area Click Select|Move pointer

Selects the pointer focus policy, either click-to-type or pointer focus. See the discussion of pointer focus policy in Chapter 1, *An Introduction to OPEN LOOK and the X Window System*.

14.1.7.7 Interface Appearance 2D|3D

Specifies whether to use the two-dimensional or three-dimensional appearance. Monochrome monitors always use 2D; color monitors default to 3D but can be controlled to 2D by this setting.

14.1.7.8 Mnemonics Off|Underline|Highlight|On-Don't' Show

AT&T-OL lets you use single-character mnemonics for many functions. This item controls whether the mnemonics will be disabled, shown by underlining or highlighting the letter, or be enabled but not displayed.

14.1.7.9 Accelerators Off|On-Show|On-Don't Show

Similarly for keyboard accelerators.

14.1.7.10 Scrollbar Placement Left|Right

Controls whether vertical scrollbars appear to the left or right of the panel they are controlling.

Don't forget to click *Apply* to make your changes take effect.

This concludes our discussion of the OPEN LOOK-specific Properties Editor or Workspace Manager. We now turn our attention to some standard X11 customization clients that will be present in any standard version of The X Window System.

14.2 xset: Setting Display and Keyboard Preferences

The *xset* client allows you to set an assortment of user preference options for the display and keyboard. Some of these are followed by `on` or `off` to set or unset the option. Note that *xset* is inconsistent with other UNIX and X programs in its use of a dash (–) as an option flag. Some options use a preceding dash to indicate that a feature be disabled; this can be confusing at first to users accustomed to seeing a dash as an introductory symbol on all options.

Although *xset* can be run any time, it is suggested that you run it at startup. These settings reset to the default values when you log out. Not all X implementations honor all of these options.

14.2.0.1 Keyboard Bell

The `b` option controls bell volume (as a percentage of its maximum), pitch (in hertz), and duration (in milliseconds). It accepts up to three numerical parameters:

```
b volume pitch duration
```

If no parameters are given, the system defaults are used. If only one parameter is given, the bell volume is set to that value. If two values are listed, the second parameter specifies the bell pitch. If three values are listed, the third one specifies the duration.

For example, the command:

```
% xset b 70 1000 100
```

sets the volume of the keyboard bell to 70 percent of the maximum, the pitch to 1000 hertz, and the duration to 100 milliseconds.

Note that bell characteristics vary with different hardware. The X server sets the characteristics of the bell as closely as it can to the user's specifications.

The `b` option also accepts the parameters `on` or `off`. If you specify `xset b on`, system defaults for volume, pitch and duration are used.

The bell can also be turned off with the option `-b`, or by setting the volume parameter to 0 (`xset b 0`).

14.2.0.2 Bug Compatibility Mode

Some X11 Release 3 clients were written to work with "features" of the MIT X11 Release 3 server, features which could more accurately be called bugs. Many of these bugs have been eliminated from the MIT Release 4 server. In order to allow certain Release 3 clients to work under the Release 4 server, the MIT X11 Release 4 server has a bug compatibility mode that can be enabled using *xset*. In this mode, the Release 4 server is compatible with Release 3 clients that depended on bugs in the Release 3 server to work properly (most notably the Release 3 version of *xterm*). This feature is hardly ever needed today; newer servers no longer provide this compatibility, and tend to ignore requests for it from *xset*.

Note that since the OpenWindows server is not derived from the MIT Sample Server, this option is **not** available under OpenWindows. However, the XWIN server is an enhanced version of the MIT server, so this option does work on the AT&T server.

To enable bug compatibility mode, use the command `xset bc`; to disable it, use the command `xset -bc`.

14.2.0.3 Keyclick Volume

The `c` option sets the volume of the keyboard's keyclick and takes the form:

```
c volume
```

volume can be a value from 0 to 100, indicating a percentage of the maximum volume. For example:

```
% xset c 75
```

sets a moderately loud keyclick. The X server sets the volume to the nearest value that the hardware can support.

The `c` option also accepts the parameters `on` or `off`. If you specify `xset c on`, the system default for volume is used.

The keyclick can also be turned off with the option `-c`, or by setting the volume parameter to 0 (`xset c 0`).

On some hardware, a volume of 0 to 50 turns the keyclick off, and a volume of 51 to 100 turns the keyclick on.

14.2.0.4 Enabling or Disabling Auto-repeat

The `r` option controls the keyboard's auto-repeat feature. (Auto-repeat causes a keystroke to be repeated over and over when the key is held down.) Use `xset r` or `xset r on` to enable key repeat. Use `xset -r` or `xset r off` to disable key repeat. On some keyboards (notably Apollo) only some keys repeat regardless of the state of this option.

14.2.0.5 Changing or Rehashing the Font Path

As discussed in Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, when a client is to be displayed in a particular font, the server by default looks for the font in several subdirectories of */usr/lib/X11/fonts*, such as *misc*, *75dpi*, and *100dpi*.

The `fp` (font path) option of *xset* can be used to change the font path, i.e., to direct the X server to search other directories for fonts called by a client. The option must be followed by a directory or a comma-separated list of directories, as in this example:

```
% xset fp /work/andy/fonts,/usr/lib/X11/newfonts
```

To restore the server-specific default font path, type:

```
% xset fp default
```

The `fp` option with the `rehash` parameter causes the server to reread the *fonts.dir* and *fonts.alias* files in each directory in the current font path. You need to do this every time you edit an alias file to make the server aware of the changes. To make the server aware of aliases, type:

```
% xset fp rehash
```

You also have to do this if you add or remove fonts. See the *xset* manual page for more information.

14.2.0.6 Keyboard LEDs

The `led` option controls the enabling or disabling of one or e fall of the keyboard's LEDs. It accepts the parameters `on` or `off` to enable or disable all of the LEDs. A preceding dash also disables all of the LEDs (`-led`).

You can also enable or disable individual LEDs by supplying a numerical parameter (a value between 1 and 32) that corresponds to a particular LED. The `led` option followed by a numerical parameter enables that LED. The `led` option preceded by a dash and followed by a numerical parameter disables that LED. For example:

```
% xset led 3
```

would enable LED #3, while:

```
% xset -led 3
```

would disable LED #3.

Note that the particular LED values may refer to different LEDs on different hardware.

14.2.0.7 Pointer Acceleration

The `m` (mouse) option controls the rate at which the mouse or pointer moves across the screen. This option takes two parameters: ***acceleration*** and ***threshold***. They must be positive integers. (The acceleration can also be written as a numerator/denominator combination separated by a slash, for example, 5/4.)

The mouse or pointer moves ***acceleration*** times as fast when it travels more than the ***threshold*** number of pixels in a short time. This way, the pointer can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen by a flick of the wrist when desired. If only one parameter is given, it is interpreted as the acceleration.

For example, the command:

```
% xset m 5 10
```

sets the pointer movement so that if you move the pointer more than 10 pixels, the pointer cursor moves five times as many pixels on the screen as you moved the pointer on the pad.

If no parameter or the value `default` is used, the system defaults will be set.

If you want to change the threshold and leave the acceleration unchanged, enter the value `default` for acceleration.

14.2.0.8 Screen Saver

X supports a screen saver to blank or randomly change the screen when the system is left unattended for an extended period. This avoids the "burn in" that can occur when the same image is displayed on the screen for a long time. The `s` (screen saver) option to *xset* determines how long the server must be inactive before the screen saver is started.

The `s` option takes two parameters: ***time*** and ***cycle***. The screen goes blank if the server has not received any input for the time interval specified by the ***time*** parameter. The contents of the screen reappear upon receipt of any input. If the display is not capable of blanking the screen, then the screen is shifted a pixel in a random direction at time intervals set by the ***cycle*** parameter. The parameters are specified in seconds.

For example, the command:

```
% xset s 600
```

sets the length of time before the screen saver is invoked to 600 seconds (10 minutes).

For a display not capable of blanking the screen, the command:

```
% xset s 600 10
```

sets the length of time before the screen saver is invoked to 10 minutes and shifts the screen every 10 seconds thereafter, until input is received.

The `s` option also takes the parameters:

`default` Resets the screen save option to the default.

`blank` Turns on blanking and overrides any previous settings.

`noblank` Displays a background pattern rather than blanking the screen; overrides any previous settings.

`off` Turns off the screen saver option and overrides any previous settings.

`expose` Allows window exposures (the server can discard window contents).

`noexpose` Disables screen saver unless the server can regenerate the screens without causing exposure events (i.e., without forcing the applications to regenerate their own windows).

OpenWindows Option—Sun users can bypass this, and instead start the program */usr/bin/screenblank* from */etc/rc.local* during system boot. This screen blank program has the advantage over the X server's that it works all the time, not just while X11 (OpenWindows) is running. For example, it will work before you have logged in, or while you are logged in but not running any window system.

14.2.0.9 Color Definition

On color displays, every time a client requests a private read/write colorcell, a new color definition is entered in the display's colormap. The `p` option sets one of these colormap entries even though they are supposed to be private. The parameters are a positive integer identifying a cell in the colormap to be changed and a color name:

```
p entry_number color_name
```

The root window colors can be changed on some servers using *xsetroot*. An error results if the map entry is a read-only color.

For example, the command:

```
% xset p 3 blue
```

sets the third cell in the colormap to the color blue but only if some client has allocated this cell read/write.

The client that allocated the cell is likely to change it again sometime after you try to set it, since this is the usual procedure for allocating a read/write cell.

14.2.0.10 Help with xset Options

The `q` option lists the current values of all *xset* preferences.

14.3 xsetroot: Setting Root Window Characteristics

You can use the *xsetroot* client to tailor the appearance of the background (root) window on a display running X.

The *xsetroot* client is primarily used to specify the root window pattern: as a plaid-like grid, tiled gray pattern, solid color, or a bitmap. You can also specify foreground and background colors (defaults are black and white), reverse video, and set the shape of the pointer when it's in the root window.

If no options are specified, or the `-def` option is specified, *xsetroot* resets the root window to its default state, a gray mesh pattern, and resets the pointer to the hollow X pointer. The `-def` option can also be specified with other options; those characteristics that are not set by other options are reset to the defaults.

Although *xsetroot* can be run at any time, it is suggested that you run it from a startup shell script, as described at the end of this chapter. All settings reset to the default values when you log out.

For a complete list of options, see the *xsetroot* reference page in Part Three of this guide. Not all X implementations are guaranteed to support all of these options. Some of the options may not work on certain hardware devices.

The `-help` option prints all the *xsetroot* options to standard output. The options you'll probably use most frequently are explained in the next section. Since only one type of background pattern can be specified at a time, the `-solid`, `-gray`, `-grey`, `-bitmap` and `-mod` options are mutually exclusive.

14.3.1 Setting Root Window Patterns

The default root window pattern is called a "gray mesh." On most displays, it is fairly dark.

The *xsetroot* client allows you to specify an alternative gray background with the `-grey` (or `-gray`) option. This tiled gray pattern is slightly lighter than the default gray mesh pattern.

The *xsetroot* client also allows you to create a root window made up of repeated "tiles" of a particular bitmap, using the option:

```
-bitmap filename
```

where ***filename*** is the bitmap file to be used as the window pattern.

You can choose any of the bitmaps in the directory */usr/include/X11/bitmaps* or make your own bitmap files using the *iconedit* or *bitmap* client (see Chapter 9, *Graphics Clients*).For example, the command:

```
% xsetroot -bitmap /usr/openwin/share/include/Xol/bitmaps/gumby \
    -fg white -bg skyblue
```

fills the root window with a tiling of the bitmap */usr/openwin/share/include/Xol bitmaps/skyscene,* a tranquil scene, using the colors white and blue.

The `-mod` option sets a plaid-like grid pattern on the root window. You specify the horizontal (x) and vertical (y) dimensions in pixels of each square in the grid. The syntax of the option is:

```
-mod x y
```

where the parameters ***x*** and ***y*** are integers ranging from 1 to 16 (pixels). (Zero and negative numbers are taken as 1.)

The larger the x and y values you specify, the larger (and more visible) each square on the root window grid pattern. Try the command:

```
% xsetroot -mod 16 16
```

for the largest possible grid squares. Then test different x and y specifications.

The *xsetroot* option:

```
-solid color
```

sets the color of the root window to a solid color. This can be a color from the color name database or a more exact color name specified by its RGB value.

The command:

```
% xsetroot -solid lightblue
```

sets the color of the root window to light blue.* See Chapter 11, *Command Line Options*, and Chapter 12, *Setting Resources*, for more information on how to specify colors.†

While this behavior may seem to be a serious bug, it is actually an optimization designed to ensure applications don't run out of colors unnecessarily. Free colormap cells can be a scarce resource. See Volume One, *Xlib Programming Manual*, for more information.

14.3.2 Foreground Color, Background Color, and Reverse Video

In addition to specifying a solid color for the root window pattern, *xsetroot* allows you to specify foreground and background colors if you set the pattern with `-bitmap` or `-mod`. The standard Toolkit options are used to set foreground and background colors: `-fg` and `-bg`. The defaults are black and white.

Colors can be specified as names from the color name database, or as RGB values. See Chapter 11, *Command Line Options*, and Chapter 12, *Setting Resources*, for more instructions on how to specify color.

If you specify reverse video (`-rv`), the foreground and background colors are reversed.

† For technical reasons, colors set with `xsetroot -solid` may change unexpectedly. When you set a color with the `-solid` option to *xsetroot*, the client allocates a colorcell, sets the color, and deallocates the colorcell. The root window changes to that color. If another client is started that sets a new color, it allocates the next available colorcell—which may be the same one xsetroot just deallocated. This results in that color changing to the new color. The root window also changes to the new color. If this happens, you can run *xsetroot* again and if there are other colorcells available, the root window changes to the new color. If all colorcells are allocated, any call to change a colorcell results in an error message.

Foreground and background colors also take effect when you set the root window pointer, as described in the next section.

Another use of *xsetroot* is to notify you when a long-running program is done. For example, if you expect a program compilation being run by *make* to take a while, you can "type ahead" an *xsetroot* command. For example:

```
make
xsetroot -solid pink
```

then iconify the window. It will run the *make*, then change your background window pink—a change you are likely to notice.

14.3.3 Changing the Root Window Pointer

By default, the pointer is an X when it's in the root window. You can change the shape of the root window pointer to one of the standard X cursor shapes or to any bitmap, using these options:

```
-cursor_name standard_cursor_name
-cursor cursorfile maskfile
```

Available as of MIT X11 Release 4, the first option allows you to set the root window pointer to one of the standard cursor symbols, which are generally listed in the file */usr/include/X11/cursorfont.h*. We've provided a list of the standard cursors in Appendix D, *Standard Cursors - X11 and OPEN LOOK*. To specify a standard cursor on a command line or in a resource file, strip the `XC_` prefix from the name. Thus, to set the root window pointer to the pirate cursor symbol, you would enter:

```
% xsetroot -cursor_name pirate
```

This second option is intended to allow you to set the root window pointer to a bitmap, perhaps one you create. The parameters ***cursorfile*** and ***maskfile*** are bitmaps. The ***cursorfile*** sets the bitmap for the pointer shape. In effect, the ***maskfile*** is placed behind the ***cursorfile*** bitmap to set it off from the root window. The ***maskfile*** should be the same shape as the ***cursorfile*** but should generally be at least one pixel wider in all directions.†

With the *xsetroot* defaults, you can observe the effect of a mask. When you move the X pointer onto the dark gray root window, the X should have a very thin white border, which enables you to see it more clearly.

For the ***cursorfile***, you can use any of the standard bitmaps in */usr/include/X11/bitmaps* or you can make your own with the *bitmap* client (see Chapter 9, *Graphics Clients*).

Every standard cursor has an associated mask. Pictures of the cursors appear in Appendix D, *Standard Cursors - X11 and OPEN LOOK*. To get an idea of what masks look like, display the cursor font using the command:

† Technically speaking, the mask determines the pixels on the screen that are disturbed by the cursor. It functions as a sort of outliner or highlighter for the cursor shape. The mask appears as a white (or background color) border around the cursor (black or another foreground color), making it visible over any root window pattern. This is especially important when a black cursor appears on a black root window.

```
% xfd -fn cursor.
```

If you are using your own bitmap as the **cursorfile**, until you get used to the way masks work, create a **maskfile** that is a copy of the **cursorfile** with all bits set, i.e., the **maskfile** should be all black* (or the foreground color). Then edit the **maskfile** to make it wider than the **cursorfile** by at least one pixel in all directions.†

To specify a root window pointer made from the smiling Gumby bitmap we created for Figure 7-2, first copy the bitmap to make a mask file:

```
% cp gumby gumby.mask
```

Then edit the *gumby.mask* file using the *bitmap* client, setting all squares inside the Gumby. (You can use the *bitmap* command box **Flood Fill** to set all the empty squares at once.) Continue to edit the bitmap, making it one pixel wider in all directions.

Then specify the new pointer with *xsetroot*:

```
% xsetroot -cursor gumby gumby.mask
```

See Chapter 9, *Graphics Clients*, for more information on using *bitmap*.

14.4 xmodmap: Modifier Key and Pointer Customization

The *xmodmap* client is used to assign (or map) key functions to physical keys on the keyboard. Primarily, *xmodmap* is used to assign so-called "modifier" key functions to physical keys but it can also change the way other keys (and even pointer buttons) function.

As described in , keys with labels such as Shift, Control, Caps Lock, etc. are called "modifier" keys because they modify the action of other keys. The number and names of modifier keys differ from workstation to workstation. Every keyboard is likely to have a Shift, Caps Lock, and Control key but after that, the babble begins. One workstation might have an Alt key, another might have a Funct key, and yet another a Gold key. On the Sun-3 keyboard, there are no less than three additional modifier keys, labeled Alternate, Right, and Left.

Because of the differences between keyboards, X programs are designed to work with *logical* modifier keynames. The logical keynames represent functions recognized by X programs. These modifier keynames can be mapped by the user to any physical key on the keyboard with the *xmodmap* client.

The logical keynames that X recognizes are:

- Shift
- Lock

† Don't be confused by the idea of a black cursor with a black mask on a black root window. Remember, the mask determines the pixels that are disturbed by the cursor—in effect creating an outline around the cursor. The outline appears in white (or specified background color), regardless of the color of the **maskfile**.

- Control
- Mod1 (Meta or Alt)
- Mod2
- Mod3
- Mod4
- Mod5

These keynames are case-insensitive.

Of these X modifier keys, only Shift, Caps Lock, Control, and Meta are in common use. Note that *uwm* also recognizes the mod keys simply by number alone (1-5) and recognizes mod1 as Meta (i.e., mod1, Meta and 1 are equivalent).

The primary function of *xmodmap* is to allow you to assign these important modifier keyname functions (Shift, Control, Meta, etc.) to convenient keys on the keyboard. For example, you could choose to map the Shift function to a single key called "Shift," to two "Shift" keys (one on either side of the keypad), to an "Alt" key, or to any other convenient key or keys on the physical keyboard. A left-handed person might choose to map modifier keys that more often are found on the left side, such as Control, on the right side of the keyboard.

In practical terms, each server will have a default keyboard configuration. The Shift, Caps Lock, and Control modifier keynames will be mapped to obvious keys. The assignment of the Meta key might be less obvious.

The *xmodmap* client allows you to print out the current assignments of modifier keyname functions to physical keys and/or to change the assignments.

xmodmap also has two other functions that you will probably use less frequently. In addition to mapping modifier keyname functions to physical keys, *xmodmap* also allows you to assign the function of *any* key on the keyboard to any other key. For instance, you can make the Backspace key and the Delete key both function as Delete keys. (This may be helpful if the Backspace key is easier to reach.)

Also, in addition to keyboard mappings, *xmodmap* can be used to display or change the pointer button assignments. Many X clients recognize logical pointer button commands. For example, holding down and dragging the first logical pointer button in an *xterm* window copies the text into memory. (In many default pointer maps, the first logical button is the leftmost button, designed to be pressed by the right index finger.) Each logical button is associated with a *button code*. The first logical button generates button code 1, the second logical button generates button code 2, etc. *xmodmap* allows you to reassign logical buttons to different physical buttons on the pointer.

Thus, basically, *xmodmap* can perform three types of mappings:

1. Assign modifier keyname functions (such as Shift, Control, Meta) recognized by X to physical keys.
2. Make any key on the keyboard function as any other key (for example, making Backspace function like Delete).
3. Reassign logical pointer button functions to other physical buttons (for example, making the rightmost physical button function as the first logical button).

In the following sections, we discuss key mapping, with an emphasis on the first type of mapping, of modifier keyname functions. Chances are, you'll have relatively little call to map other key functions (such as Backspace), though we have included an example of one such mapping, just in case.

After considering key mapping, we'll take a look at the much simpler issues involved in mapping pointer button functions. As you might expect, when you're changing the functionality of (up to) three pointer buttons, it's fairly simple to keep track of what you're doing.

On the other hand, mapping modifier key functions to physical keys can be more than a little confusing. In order to understand the mechanics of mapping keys, we first need to take a look at some terms used to describe keyboard keys.

14.4.1 Keycodes and Keysyms

Each key on a physical keyboard can be identified by a number known as a *keycode*. (Technically speaking, a keycode is the actual value that the key generates.) Keycodes cannot be mapped to other keys. No matter what functions you assign to various keys with *xmodmap*, the keycode associated with each physical key remains the same.

In addition to a keycode, each physical key is associated with a name known as a keysym. A *keysym* (*key sym*bol name) is a name that represents the label on a key (theoretically) and corresponds to its function.

Alphanumeric keys generally have obvious keysyms, corresponding to the label on the key: for example, the keysym for the key labeled "H" is *h*. Unfortunately, a keysym does not always correspond to the key label. For example, on a Sun-3 workstation, though the keysym for the key labeled "Return" is *Return*, the keysym for the key labeled "Alternate" is *Break*, and the keysym for the key labeled "Right" is *Meta_R*.

While each keycode is tied to a physical key, each keysym corresponds to a *function* —and the keysym/function is mapped to a particular physical key (keycode). Every keyboard has a default assignment of keysyms to keycodes. In most cases, each physical key on the keyboard will be associated with a different keysym. As we'll see, however, the keysym (function) associated with a particular physical key (keycode) can be changed. This is done by assigning the keysym of one key to the keycode of another.

The modifier keynames recognized by X are not to be confused with keysyms. The X modifier keys are limited to the eight keynames discussed previously and are assigned *in addition* to the regular keysym/keycode pairings. In other words, when a physical key is mapped to function as the X Control key, it already has a default functionality (keysym) and keycode.

By default, most modifier keyname functions are mapped to keys having keysyms representing the same function. For example, the X Control keyname is probably mapped to the key labeled Control, and having the keysym Control.

The Meta modifier keyname is probably also assigned to a key having the keysym Meta. However, determining which physical key has the keysym Meta can be something of a puzzle. Later in this chapter, we'll consider a program called *xev*, which can be used to determine the keysym and keycode of any physical key.

With this background information in mind, we can now tackle a procedure to map modifier keynames.

14.4.2 Procedure to Map Modifier Keys

In order to change modifier key mappings with a minimum of confusion, you should perform these steps:

1. Display the current *modifier* key mappings using *xmodmap*.
2. Then print out the default assignments of keysyms to keycodes for *all* keys, using *xmodmap* with the `-pk` option. Save this list of the default key assignments in a file as a reference.
3. Experiment with the *xev* client to determine the keysyms associated with certain physical keys. This will help you find the key(s) assigned as the Meta modifier key (which probably also has the keysym Meta).
4. Once you're familiar with the current assignments, you can remap modifier keys using *xmodmap*.

14.4.3 Displaying the Current Modifier Key Map

Before mapping any modifier keynames, you should take a look at the current assignments. With no options, *xmodmap* displays the current map of X modifier keynames to actual keys. Type *xmodmap* and you get a display similar to this:

```
xmodmap: up to 3 keys per modifier, (keycodes in parentheses):

shift Shift_L (0x6a), Shift_R (0x75)
lock Caps_Lock (0x7e)
control Control_L (0x53)
mod1 Meta_L (0x7f), Meta_R (0x81)
mod2 Mode_switch (0x14)
mod3 Num_Lock (0x69)
mod4 Alt_L (0x1a)
mod5 F13 (0x20), F18 (0x50), F20 (0x68)
```

For each logical keyname (on the left), *xmodmap* lists one or more keysyms, each followed in parentheses by an actual hardware keycode. The keycodes displayed by *xmodmap* are represented in hex. As we'll see, the equivalent decimal and octal keycodes are also accepted as arguments to *xmodmap*.

Table 14-3. Arguments to xmodmap

Logical modifier keyname recognized by X	**Keysym**	**Keycode (hex version)**
Shift	Shift_L Shift_R	(0x6a) (0x75)
Lock	Caps_Lock	(0x7e)
Control	Control_L	(0x53)
Mod1	Meta_L Meta_R	(0x7f) (0x81)

In this mapping, two keys are assigned as Meta (mod1) keys: keys having the keysyms Meta_L and Meta_R (for left and right, apparently one on each side of the keyboard). Unfortunately, as you can see, this doesn't really tell you which keys these are on the physical keyboard. You still need to know which physical keys (keycodes) have the keysyms Meta_L and Meta_R. You can determine this using the *xev* client, described later in this chapter.

14.4.4 Determining the Default Key Mappings

Before you start mapping keys, you should display and save a map of the default assignments of keysyms to keycodes. Running *xmodmap* with the `-pk` option prints a current map of all keyboard keys to standard output. This map, called a keymap table, lists the decimal keycode on the left and the associated keysym(s) on the right. Figure 14-6 shows a portion of a typical keymap table, for a Sun-4 keyboard.

Notice that each keysym is listed by a keysym name (comma, Caps_Lock, etc.) and a keysym value (0x002c, 0xffe5, etc). For our purposes, this value is irrelevant. It cannot be supplied as a keysym argument to *xmodmap*.

As you can see, the keymap table lists regular keyboard keys (C, V, comma, slash, space, etc.), and function/numeric keypad keys (R13, F35, etc.) as well as modifier keys (Caps_Lock, Meta_L and Meta_R). If you map several keys, you may get confused as to the original assignments. Before you map any keys, we suggest you redirect the keymap table to a file to save and use as a reference:

```
% xmodmap -pk > keytable
```

The keysyms recognized by your server are a subset of a far greater number of keysyms recognized internationally. The file */usr/include/X11/keysym.h* lists the keysym *families* that are enabled for your server. The file */usr/include/X11/keysymdef.h* lists the keysyms in each of the families enabled for your server, as well as the keysyms in several other families. See Appendix H, *Keysyms*, of Volume Two, *Xlib Reference Manual*, for more information on keysyms and tables of the most common ones.

```
% xmodmap -pk
There are 4 KeySyms per KeyCode; KeyCodes range from 8 to 132.

 KeyCode Keysym (Keysym) ...
 Value Value (Name) ...

 109 0x0043 (C)
 110 0x0056 (V)
 111 0x0042 (B)
 112 0x004e (N)
 113 0x004d (M)
 114 0x002c (comma) 0x003c (less)
 115 0x002e (period) 0x003e (greater)
 116 0x002f (slash) 0x003f (question)
 117 0xffe2 (Shift_R)
 118 0xff0a (Linefeed)
 119 0xffde (R13) 0xffde (R13) 0xffb1 (KP_1) 0xff57 (End)
 120 0xff54 (Down) 0xffdf (F34) 0xffb2 (KP_2)
 121 0xffe0 (F35) 0xffe0 (F35) 0xffb3 (KP_3) 0xff56 (Next)
  . . .
 125 0xff6a (Help)
 126 0xffe5 (Caps_Lock)
 127 0xffe7 (Meta_L)
 128 0x0020 (space)
 129 0xffe8 (Meta_R)
```

Figure 14-6. Partial keymap table

14.4.5 Matching Keysyms with Physical Keys Using xev

The keysym and keycode for any key can be determined with the *xev* client.†

If you cannot use *xev*, you must rely on the keymap table and a little deductive reasoning. Since certain OPEN LOOK functions have keyboard shortcuts involving the Alt (Meta) key, at least on the AT&T version of OPEN LOOK, testing these shortcuts should help you locate this key. See , for more information.

This is particularly useful for finding the Meta key(s). The *xev* client is used to keep track of *events*, packets of information that are generated by the server when actions occur and are interpreted by other clients. Moving the pointer or pressing a keyboard key cause input events to occur. (For more information about events, see Volume One, *Xlib Programming Manual*.)

To use *xev*, enter the command:

```
% xev
```

† *xev* is an MIT X11 Release 3 standard client. In Release 4, it has been moved to the *demos* directory; on Open-Windows it is in */usr/openwin/demo/xev*. If an executable version does not exist on your system, ask your system administrator.

in an *xterm* window, and then use the pointer to place the *xev* window, as in Figure 14-7.

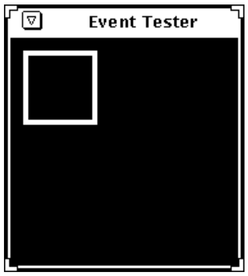

Figure 14-7. xev window

Within the *xev* window is a small box. Move the pointer inside this box. When you type a key inside the box, information about the key, including its keysym and keycode, will be displayed in the *xterm* window from which you started *xev*. The relevant information will look like this:

```
. . . keycode 127 (keysym 0xffe7, Meta_L) . . .
```

Notice that the keycode is given as a decimal number. You can use the decimal keycode as an argument to *xmodmap*. The keysym is listed by name, Meta_L, and value, 0xffe7. Again, this value cannot be supplied as a keysym argument to *xmodmap*. (See the *xev* reference page in Part Three of this guide for more information.)

To find the Meta key, type a few likely keys in the *xev* window. To terminate the program, you can use the *Window* menu or (if you didn't background the *xev*) just type Control-C in the window from which you invoked *xev*.

14.4.6 Changing the Map with xmodmap

xmodmap executes an expression or list of expressions that is interpreted as instructions to modify the key (or pointer) map. The expressions that can be interpreted by *xmodmap* are described in the next section.

xmodmap has this syntax:

xmodmap [***options*** [***filename***

An expression can be executed in either one of two ways:

- From the command line, using the `-e` ***expression*** option. This option specifies an expression to be executed (as an instruction to modify the map). Any number of expressions may be specified from the command line. An ***expression*** should be enclosed in quotes.
- Entered in a file that is used as an argument to *xmodmap*. Several expressions can be entered in one file.

See the *xmodmap* reference page in Part Three of this guide for a complete list of options. Other than `-e` ***`expression`***, the most important options for our purposes are listed below.

`-n` Indicates that *xmodmap* should not change the key mappings as specified in the ***`filename`*** or command line expression but should display what it would do. A handy test. (Only works with key mappings, not with expressions that change the pointer map.)

`-verbose` Indicates that *xmodmap* should print information as it parses its input.

`filename` specifies a file containing *xmodmap* expressions to be executed (as instructions to modify the map). This file is usually kept in the user's home directory with a name like *.xmodmaprc*.

14.4.6.1 Expressions to Change the Key Map

The expressions interpreted by *xmodmap* can be used to perform these types of key mappings:†

1. Assign and remove keysyms as modifier keynames recognized by X.
2. Map any keysym (function) to any physical key (keycode).

This list shows allowable expressions, divided by function. (Using *xmodmap* with the `-grammar` option returns a help message with much of this information.) Those expressions that include an equal sign require a space before and after the sign.

1. To assign and remove keysyms as modifier keynames:

`clear` ***MODIFIERNAME***

Removes all entries in the modifier map for the given modifier, where valid modifier names are: shift, lock, control, mod1, mod2, mod3, mod4, and mod5 (case does not matter in modifier names, although it does matter for all other names). For example, the expression `clear Lock` will remove all keys that were bound to the lock modifier.

`add` ***MODIFIERNAME=KEYSYMNAME***

Adds the given keysym to the indicated modifier map. For example, you could make the Alt key an additional shift modifier key. The keysym name is evaluated after all input expressions are read to make it easy to write expressions to swap keys.

`remove` ***MODIFIERNAME=KEYSYMNAME***

Removes the given keysym from the indicated modifier map (unmaps it). For example, remove Caps_Lock as the lock modifier key. Unlike with the `add` expression, the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether they have been reassigned.

† Expressions to change the pointer map are discussed in the section "Displaying and Changing the Pointer Map," later in this chapter.

2. To map any keysym(s) to any physical key (keycode):

`keycode` ***NUMBER=KEYSYMNAME***

Assigns the keysym to the indicated keycode (which may be specified in decimal, hex or octal). Usually only one keysym is assigned to a given code.

`keysym` ***KEYSYMNAME=KEYSYMNAME***

Assigns the keysym on the right to the keycode of the keysym on the left. Note that if you have the same keysym bound to multiple keys, this might not work.

14.4.6.2 Key Mapping Examples

Expressions can be used on the *xmodmap* command line or entered in a file that is then used as an argument to *xmodmap*. Note that *xmodmap* should be run from your startup script (discussed in) to take effect for all clients in the login session. This section includes three examples, corresponding to the three types of mappings you can perform.

Remember that including the `-n` option on the *xmodmap* command line allows you to see what the new mappings *would* be, without actually performing them. This can be very useful, particularly while you're learning to use *xmodmap* and getting used to the syntax of expressions. (Note, however, that `-n` cannot be used with expressions to change the pointer mapping.)

First, the *xmodmap* client also allows you to assign logical modifier keynames to physical keys. A not so obvious feature of *xmodmap* is that to change the mapping of a modifier key, you must first remove that key from the current modifier map. For example, to swap the left Control and (Caps) Lock keys, you would first need to unmap both physical keys (Caps_Lock, Control_L) from their respective modifier keynames (lock, control):

```
remove lock = Caps_Lock remove control = Control_L
```

And then reverse the mappings:

```
add lock = Control_L add control = Caps_Lock
```

If you then type *xmodmap* without options, you see the new map:

```
xmodmap: up to 2 keys per modifier, (keycodes in parentheses):
shift Shift_L (0x6a), Shift_R (0x75)
lock Control_L (0x53)
control Caps_Lock (0x7e)
mod1 Meta_L (0x7f), Meta_R (0x81)
mod2
mod3
mod4
mod5
```

The key with the keysym Control_L functions as a Lock key and the key with the keysym Caps_Lock functions as a Control key.

Second, *xmodmap* allows you to assign any keysym to any other key. Here is some code that sets up the OpenWindows key assignments:

```
xmodmap -e 'keysym F1 = Help' \
    -e 'add mod1 = Meta_L Meta_R'
```

This sets F1 to be the OPEN LOOK Help key, and the Left and Right keys to be Meta (or ALT) keys. This example is excerpted from the Sun startup scripts in */usr/openwin/lib*. As another example, you might make the Backspace key function as a Delete key:

```
% xmodmap -e 'keysym BackSpace = Delete'
```

Then when you display the keymap table and *grep* for the Delete keysym, you'll see that it is assigned twice. On the command line of an *xterm* window, type:

```
% xmodmap -pk | grep Delete
```

and you'll get two lines from the current keymap table, similar to these:

```
50 0xffff (Delete)
73 0xffff (Delete)
```

The 50 and 73 are keycodes representing two physical keys. As you can see, both of these keys now function as Delete keys.

This example suggests some of the confusion you can experience using *xmodmap*. We know that one of these keys previously functioned as the Backspace key. But how can we tell which one? Here is an instance when our default keymap table comes in handy. If you've run `xmodmap -pk` and redirected it to a file before changing any mappings, you can check the file for the keysyms originally associated with the keycodes 50 and 73. In this case, the file tells us 50 originally was Backspace and 73 was Delete.

Of course, you could also figure out the original assignments by remapping one of the keycodes to Backspace. Then, if the key marked Backspace functions as marked, you know you've mapped the keysym to the original keycode. But, as you can see, the default keymap table can greatly simplify matters.

This example also implies that there are advantages to using expressions of the form:

```
keycode number = keysymname
```

This expression syntax requires you to be aware of default keycode/keysym assignments. Also, if you explicitly assign a keysym to a particular keycode, it's much easier to keep track of what you're doing and retrace your steps if necessary. On the down side, though keysyms are portable, keycodes may vary from server to server. Thus, expressions using this syntax cannot be ported to other systems.

14.4.7 Displaying and Changing the Pointer Map

If you want to change the assignment of logical pointer buttons to physical buttons, you should first display the current pointer map with the `-pp` option to *xmodmap*. A typical pointer map appears in Figure 14-8.

```
There are 3 pointer buttons defined.
Physical Button
Button   Code
    1    1
    2    2
    3    3
```

Figure 14-8. Pointer map

This is a fairly simple map: the physical buttons are listed on the left and the corresponding logical functions (button codes) are listed on the right.

These are typical assignments for a right-handed person: the first logical button is the left-most button, designed to be pressed by the right index finger. The *xmodmap* client allows you to reassign logical buttons so that the pointer can be more easily used with the left hand.

The *xmodmap* client allows you to change the pointer map.* There are two *xmodmap* expressions: one to assign logical pointer buttons (button codes) to physical buttons; and another to restore the default assignments. The syntax of the expressions is:

`pointer=`***`x y z`***

Sets the first, second, and third physical buttons to the button codes ***x***, ***y***, and ***z***.

`pointer=default`

Sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a code of 2, etc.).

Being able to change the pointer button assignments is very useful if you happen to be left-handed and would like the rightmost physical button to function as the first logical button (i.e., generate button code 1). To configure the pointer for a southpaw:†

```
% xmodmap -e 'pointer = 3 2 1'
```

If you then display the pointer mappings with `xmodmap -pp`, you get this:

```
There are 3 pointer buttons defined.
Physical Button
Button Code
1 3
2 2
3 1
```

You can then push the first logical button (button code 1, which OPEN LOOK uses for SELECT) with the index finger of your left hand; button code 2, ADJUST, with the second finger, and button code 3, MENU, with the third finger.‡

You can return to the default pointer button assignments by entering:

```
% xmodmap -e 'pointer = default'
```

Note that AT&T-OL's version of the Properties Editor lets you assign the pointer buttons more easily, without using *xmodmap*.

† Remember that the `-n` option, which allows you to see what *xmodmap would* do without performing the changes cannot be used with expressions to change the pointer mapping.

‡ Users of OpenWindows may need to take additional action to convince NeWS clients to switch the pointer buttons around.

14.5 xkeycaps - visual keyboard mapping

Maybe Lee can provide this?