- Click the first pointer button on either the up or down arrow (the slider in the first vertical bar echoes this motion). This action causes the color area to display the next shade above or below the current shade in the colormap. (If you press and hold down the pointer button, the motion is continuous—you can browse several adjacent shades in the colormap this way.)
- Instead of using the arrow keys, use the slider in the first vertical bar to browse adjacent shades. The slider operates in the same way the *xterm* scrollbar's "thumb" does. (If you're not familiar with the scrollbar's operation, see Chapter 5.)
- Use the Hue Bar. See the next section for instructions.
- Enter alternate numeric values in the three small text windows to the right of the color area. (These Text widgets operate as described under "The xedit Text Editor" earlier in this chapter.) To enter meaningful values that approximate what you want, you may need to understand the science of color a bit more. Chapter 12 should be helpful in this regard. The section "Working with the Numeric Color Values" (later in this chapter) also provides some insights.

The first method is the simplest, but it is also the least precise. It's particularly difficult to click on the hue you want in the second vertical bar, which represents the adjacent hues using very narrow bands of color. It's much easier to click on the shade you want in the fourth vertical bar (with the liability that the spectrum is more limited).

In our example, the current color (sky blue) is represented by the second shade from the bottom on the fourth vertical bar. The lines in the third bar—as well as the number 2 in the color area—indicate this fact. (Though it appears that the current color is the third block from the bottom of the fourth vertical bar, the first block is actually blank—not a part of the colormap.) Let's select another color for the color area by clicking the first pointer button seven shades higher on the fourth vertical bar (a darker blue). Figure 8-22 shows the resulting *xtici* window.

Notice that the number in the color area has been changed to 9, to indicate the ninth shade in the colormap. The slider in the first vertical bar has also moved. The numbers for hue, value, and chroma reflect the new shade. The other major change in the *xtici* window is the appearance of the Hue Leaf, the meaning and use of which we'll discuss later on.

Changing the Hue with the Hue Bar

The Hue Bar allows you to view the hues the Colormap Scale offers in a wider spectrum of lightness and intensity—and to choose one of these shades to edit. The Hue Bar area contains arrow keys, a vertical bar with a slider, and the actual Hue Bar itself. When you first run *xtici*, the Hue Bar is blank. To fill the bar:

1. Display the Options menu by clicking the first pointer button on the menu command box.
2. Click on the Fill Hue Bar item.

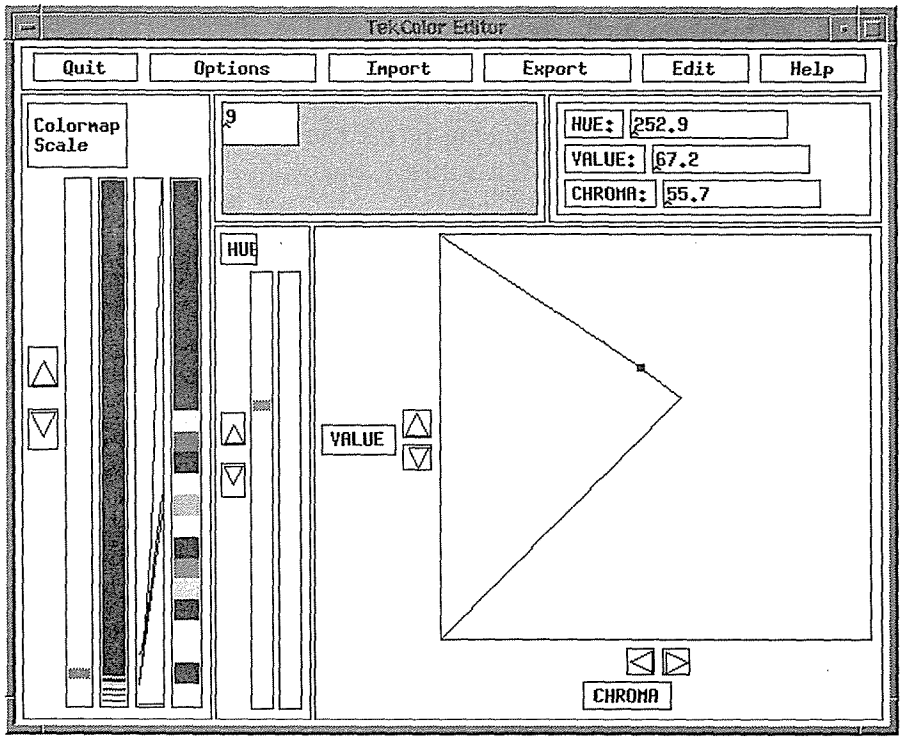

Figure 8-22. Changing the hue by clicking in the Colormap Scale area

The Hue Bar will display a range of hues. (A shorter representation of the same spectrum will also be added to the second vertical bar in the Colormap Scale area.) You can select one of the hues in the bar to edit in the color area in several ways:

- Click the first pointer button on the color you want in the Hue Bar.
- Click the first pointer button on either the up or down arrow (the slider echoes this motion). This action causes the color area to display the next shade above or below the current shade in the bar. (Press and hold down the pointer button to browse several shades.)
- Use the slider in the vertical bar to the right of the arrows to select a hue. See the instructions for using the *xterm* scrollbar's "thumb" in Chapter 5.

Note that you don't have to fill the Hue Bar in order to adjust the hue using any of these methods. (You can even click on the blank Hue Bar, though this "blind" method is not particularly desirable.) The color area and the numeric values will be updated to match the shade you choose regardless.

If you select a hue by any of the mechanisms in the Hue Bar, you can then adjust the color using the Hue Leaf or by the less intuitive method of changing the numeric values.

If you fill the Hue Bar and then try to select a hue from the Colormap Area, there may occasionally be minor problems allocating color cells for the *xtici* application. In such cases, a dialog box will request your input. See "Problems Allocating Color Cells" for more information.

Adjusting the Color with the Hue Leaf

The Hue Leaf represents a range of possibilities for the current hue. The hue can vary in value (lightness to darkness) and in chroma (the amount of the hue present; also known as saturation or intensity)—and within the Hue Leaf, it varies dramatically. For instance, the lightest possible shade of *any hue* is white; the darkest is black. (That is, the spectrum of possible values always spans white to black.) The range of chroma or saturation is more reliant on the actual hue. For example, in most cases, a red hue can exist in a wider range of intensities than a yellow hue.

The Hue Leaf is always triangular, but the shape of the triangle depends on the possibilities of varying the hue. The triangular Hue Leaf is turned on its side, so that the base is actually vertical—flush against the left side of the box containing the leaf. The range of value is represented along this vertical edge (white is at the top; black at the bottom). The range of chroma (saturation) is represented horizontally, with least to most saturation appearing from left to right. The Hue Leaf is intended to represent all possible variations of the hue in question.

When you first run *xtici*, the Hue Leaf appears blank and contains a small square dot cursor. This cursor marks the place in the leaf that corresponds to the current version of the hue. To get a better idea of the range of possibilities for the hue, you can fill the leaf:

1. Display the Options menu by clicking the first pointer button on the menu command box.
2. Click on the Fill Leaf item.

The Hue Leaf will display the range of possibilities for the current hue. Variations between shades create a sort of striped or checkerboard pattern.

The Hue Leaf allows you to fine tune the hue in question. Notice the arrow keys beside the Value and Chroma labels bordering the leaf. You can adjust the value and chroma by clicking on these arrows. For example, you would click on the up arrow next to Value to make the hue lighter (white is at the top of the value range). The dot cursor will move up within the leaf and the color area and numeric values will be updated to reflect the changes.

Click on the right arrow next to Chroma to get a more intense hue. The dot cursor will move to the right within the leaf and the color area and numeric values will be updated to reflect the changes.

As an alternative to using the arrow keys, you can use the pointer to move the square dot within the leaf. Either click on the shade you want within the leaf or hold down the pointer on the dot and drag it within the leaf.

As is the case with the Hue Bar, you don't have to fill the Hue Leaf to adjust the color using any of these methods. The color area and the numeric values will be updated regardless.

If you fill the Hue Leaf and then try to select a hue from the Colormap Area, *xtici* may have trouble allocating color cells and a dialog box with be displayed. See "Problems Allocating Color Cells" for more information.

Selecting and Pasting the Numeric Color Value

Once you have the color you want in the color area, you can select the numeric description of that color to paste on the command line, in a resource file, in a color database file, etc. To make the color value the PRIMARY text selection:

1. Display the Edit menu by clicking the first pointer button on the menu command box.
2. Click on the Copy Color –> item. A submenu is displayed.
3. Click on the format (color space) you want. The TekHVC and CIE u'v'Y items select portable color values; the RGB item selects the non-portable RGB color format.

TekHVC is a good choice. You can then paste the color value by clicking the second pointer button. For example, you might enter:

```
% xbiff -fg
```

and then click the second button to specify the color (and run the process in the background):

```
% xbiff -fg TekHVC:223.93036/72.45283/29.67013 &
```

On our display, this color value produces a deeper version of the sky blue from our original example. If you intend to use a color multiple times, it's a good idea to pair the numeric value with a name in an Xcms database.

Note that *xtici* handles RGB values in an unusual way. The window displays RGB values in decimal notation; however, if you select RGB from the Edit menu, the output is in hexadecimal notation! This can be a bit confusing, particularly if you want to place RGB values in an RGB or Xcms database. An RGB database requires decimal values; an Xcms database recognizes RGB values (among others), but they must be in hexadecimal notation. If you have the decimal numbers to input to *xtici*, the editor can in effect perform the conversion to hex; or you can use the UNIX *bc*(1) utility to convert numbers from one notation to another. See Chapter 12 for instructions on using *bc* and on creating color databases.

Working with the Numeric Color Values

We've seen several ways to edit color specifications using *xtici*'s graphic elements: bars, sliders, arrow buttons, etc. When you change a color using one of these methods, the numeric values corresponding to the color are updated dynamically. However, you can also interact with *xtici* by entering numeric values yourself.

Thus far we've only seen the default Hue, Value, and Chroma number displays. (These provide a number in the portable TekHVC color space.) But remember that *xtici* can interpret and output two additional color spaces: the portable CIE u'v'Y format and the non-portable RGB format. You can display the specification for the current color in any of these formats by using the Options menu.

1. Display the menu by clicking the first pointer button on the Options command button.
2. Click on the Coordinates –> item. A submenu is displayed revealing two options: RGB and CIE u'v'Y.

3. These menu items are toggles between the color space named and the default TekHVC color space. Thus, selecting RGB once toggles the decimal values for RED, GREEN, and BLUE. Selecting RGB a second time recalls the TekHVC values. Click on the format (color space) you want.

Let's consider a couple of ways you might work with *xtici* using numbers. Keep in mind that all of the numeric values are contained in small text windows. Use the editing commands described under "The xedit Text Editor" (earlier in this chapter) to change the values.

Suppose you want to edit a color from the standard RGB database. To place that color in the *xtici* color area:

1. Check the RGB decimal values in *rgb.txt*. (See Chapter 12.)
2. Using the Coordinates submenu of the Options menu, toggle the RGB numeric values.
3. Place the values from *rgb.txt* in the RED, GREEN, and BLUE text windows to the right of the color area.

As soon as you move the pointer out of the text window area, a dialog box will prompt:

```
Apply last keyboard input?
```

If you've entered the correct figures, select OK and the color area will be updated; otherwise, Cancel and continue editing.

As another example, say you've created a color using another editor, such as *xcoloredit*, that outputs values in the non-portable RGB format. If you enter the decimal versions of these values in the *xtici* window (as described in the previous example), *xtici* provides the portable color space equivalents.

Problems Allocating Color Cells

Because of the nature of colormaps and the way color cells are allocated, certain problems may arise in working with *xtici*. One is a simple, albeit confusing "technicolor" effect. Depending on where the input focus is, applications may appear to swap colors and the shade in the *xtici* color area may not appear accurate. You're liable to get a more precise picture when the *xtici* window has the input focus, however.

Another potential problem: you may not be able to select a color in one area of the *xtici* window if it is being used in another area. If such a conflict arises, a dialog box will inform you. For example, say you select a color in the Colormap Area that is also being used in the Hue Bar, you may get a dialog to the effect that:

```
This color cell is used to fill the Hue Bar.
Hues will be removed to edit this cell.
```

The box provides the possible responses OK and Cancel. The safest course of action is to click on Cancel and then try to select the hue by another method. Clicking on the shade you want in the Hue Bar should work; or you might turn the Hue Bar off (the menu item is a toggle) and try to select the color by dragging the Hue Bar slider; or you could enter the appropriate numeric values, etc.

If you click on OK, the Hue Bar (and possibly the leaf and part of the Colormap Scale area) will be blanked out and the colors *xtici* is displaying will be changed. In such a case, try clicking on any visible color in the Colormap Scale area to begin editing again.

A similar conflict can arise if you select a hue in the Colormap Scale area that is also being used in the Hue Leaf:

```
This color cell is used to fill the leaf.
Fill will be removed to edit this cell.
```

Again, it's a good idea to click on Cancel and then try to select the hue by another method. Clicking on the shade you want on the Hue Bar or Leaf should work; or you might turn either or both the bar and leaf off (the menu items are toggles) and try to select the color by another method.

Selecting OK will blank out the leaf (and possibly the bar and part of the Colormap Scale area) and change the colors the *xtici* window is displaying. Again, try clicking on any visible color in the Colormap Scale area.

Various other colormap conflicts can arise. Use the dialog boxes—and your own experience—for guidance.

Quitting xtici

To quit the application, click the first pointer button on the Quit command button—the leftmost one on the menu bar.

Part Two:

Customizing X

X has been designed to put the user in the driver's seat. Everything from the colors and sizes of windows to the contents of mwm *menus can be customized by the user. This part of the book tells you how to reshape X to your liking.*

Command-line Options
Setting Resources
Specifying Color
Customizing mwm
Setup Clients

9

Working with Motif Applications

This chapter examines some of the features common to applications written with the Motif Toolkit.

In This Chapter:

9
Working with Motif Applications

The Athena widget set provides X Toolkit applications with certain common features, many of which have been described in Chapter 8. An application coded using the Motif widget set has a slightly different look and feel.

In the remainder of this chapter, we'll look at some of the features you're liable to encounter in a Motif application and learn how to use them. Some of these features are provided in a slightly different flavor by the Athena widget set; others are unique to Motif.

Many of the sample components we're using are taken from the Motif *periodic* demo program, which is a "periodic table" of the Motif widgets. You can play with *periodic* to begin learning to use many common features of Motif applications. However, since the program simply demonstrates the various widgets without actually performing any practical action, you'll probably need to use some real applications as well. If you've been running *mwm*, you already know how to use several Motif features.

The following sections mention the comparable Athena widgets where appropriate. Some of the Athena widgets are illustrated using the standard MIT clients in Chapters 5, 7, and 8.

Before examining the various Motif features, however, let's consider some basics of using the pointer with a Motif application.

Pointer Button Usage

When you're working with an application coded using the Motif toolkit, you can generally rely on the pointer buttons to work as follows:

Button one: Referred to in program internals as "BSelect" (for "Button Select"), the first button enables you to "select" or activate graphical components. For example, you would use the first button to direct focus, to select text, to respond to a dialog box, etc.

Button two: Referred to as "BTransfer" (for "Button Transfer"), the second button is used to "drag" text, graphic images, etc., from one widget and "drop" them into another widget. As we'll see later, this "drag and drop" capability is one of the major improvements of Motif 1.2.

Button three: Referred to as "BMenu" (for "Button Menu"), the third button is used to post pop-up menus. For example, you post the *mwm* Root Menu by pressing and holding the third pointer button on the root window.

Now that you understand the basics of pointer actions, let's consider the various application components you're liable to encounter.

The Periodic Table of Motif Widgets

The *periodic* demo program (pictured in Figure 9-1) provides a compact and comprehensive survey of the Motif 1.2 widget set.

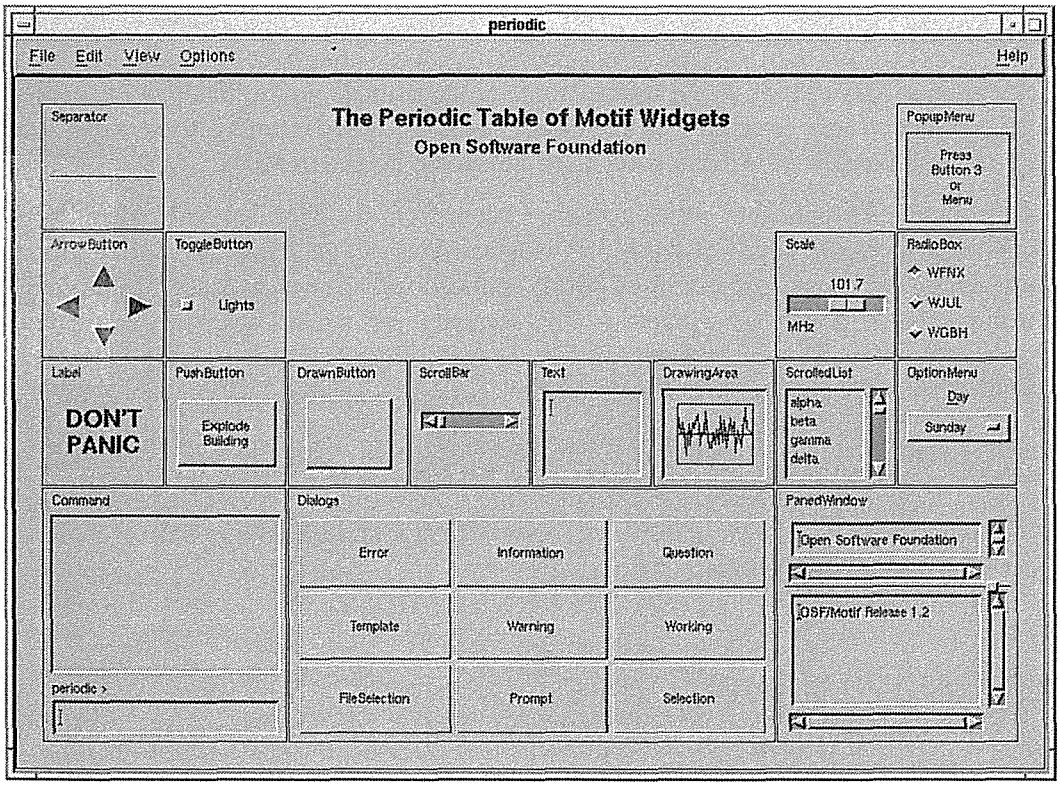

Figure 9-1. The periodic table of Motif widgets

Widgets are the building blocks of applications coded using Xt-based toolkits, such as the Motif toolkit. From a user's perspective, this definition of "widget" is not particularly significant. What is significant is that widgets create graphical elements like scrollbars, text windows, and push buttons, which you use to work with an application.

Keep in mind that some widgets don't help you do anything. For instance, the Separator widget (in the upper-left corner of the periodic table) is simply a divider line, often used below menu titles. Other widgets represented in the table are *composites*: several simpler widgets combined for a particular purpose. For example, the FileSelectionBox widget contains two text input fields, two scrollable lists (each also a composite!), and four or more push buttons, the sum total of which help you select a file from a directory hierarchy. The file selection box is a particularly complicated example. However, in most cases, when you know how to use the various components, you can deal with them in any combination.

We mention the names of the various widgets discussed in this chapter, but unless you're interested in programming, this sort of classification is probably superfluous. (If you want to set resources at the widget level, the names will be more relevant. See Appendix G, *Widget Resources*, for more information.) When you're running an application, it doesn't matter what a feature is called—only how it's used. Now let's learn how to use some of the most important Motif features. After reading this chapter and playing with a client or two, these skills will become intuitive.

Menus

Motif applications may feature three types of menus, the first two of which you have already encountered:

- Pull-down menus, such as *mwm*'s Window Menu.
- Pop-up menus, such as *mwm*'s Root Menu.
- Option menus.

The following sections describe these three types of menus. Keep in mind that each of these menus can also be what is known as a "tear-off" menu: that is, you can choose to post the menu and "tear off" an image of it that remains posted (in its own window) until you remove it. We'll take a closer look at tear-off menus after considering the three basic menu types.

Pull-down Menus

You display a pull-down menu from a graphical element on an application window. The *mwm* Window Menu is a pull-down, which is displayed from a button in the upper-left corner of the frame. More often, though, a pull-down is displayed from a horizontal bar known as a *menu bar*. Figure 9-2 illustrates the menu bar on the *periodic* window.

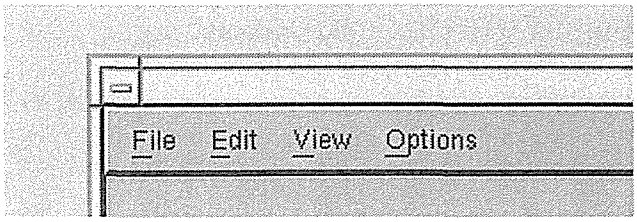

Figure 9-2. Periodic menu bar

Each word on the bar is a menu title; you display the menu by placing the pointer on its title and clicking the first pointer button. The title becomes raised and highlighted by a box, the menu is displayed and the first selectable item is also raised and boxed. Figure 9-3 shows *periodic*'s File pull-down menu.

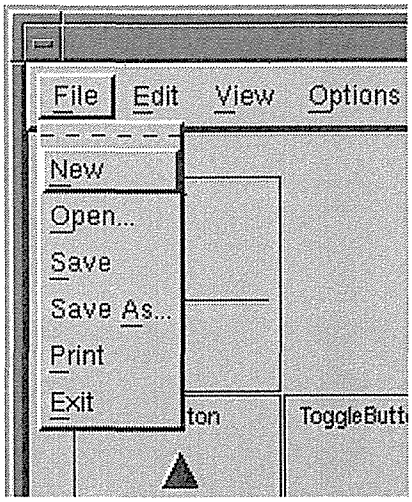

Figure 9-3. Periodic File menu

Notice that one letter of each menu item is underlined. As explained in Chapter 4, that letter represents a unique abbreviation, or mnemonic, for the menu item, which can be used to select the item.

Some menus (such as *mwm*'s Window Menu) may also provide a keyboard shortcut, or *accelerator*, for each item. These shortcuts generally appear in a right-hand column, opposite the item labels. An accelerator can be used to invoke the action without displaying the menu at all (though they also work while the menu is displayed).

When you've displayed a menu by placing the pointer on the title and clicking the first button, you can select an item by:

- Placing the pointer on the item and clicking the first button.
- Typing the mnemonic abbreviation for the menu item.

- Typing the accelerator key combination, if available. (Though these are intended to save you the trouble of displaying the menu, they also work when it is displayed.)
- To select the boxed item (the first available for selection), you can alternatively press either the Return key or the space bar.

You can also display a menu from a menu bar by placing the pointer on the title and pressing the first pointer button. The menu is displayed as long as you continue to hold the pointer button down. To select an item, drag the pointer down the menu (each item is highlighted by a box in turn), and release the button on the item you want.

Notice the apparant "perforation" below the File menu title. This dotted line indicates that the menu has "tear-off" functionality—that is, you can keep the menu displayed in its own window and access it whenever you like. We'll discuss tear-off menus a bit later.

Pop-up Menus

The *mwm* Root Menu is a typical pop-up menu. Depending on the application, you pop up a menu by placing the pointer in a particular context (that is, on a particular graphical element) and either:

- Pressing and holding the third pointer button. (This is how *mwm*'s Root Menu is displayed.)
- Clicking the third pointer button.

When you display a menu by the former method, you make a selection by dragging the pointer down the menu and releasing the button on the item you want.

When you display a menu by the latter method, you select an item by clicking on it with either the first or third pointer button. You can pop the menu down by clicking either of these buttons off the menu.

Keep in mind that pop-up menus generally provide shortcuts for functions that can be performed in other ways. Since there are no labels to indicate that a pop-up menu exists, you'll have to rely on the individual program documentation.

Option Menus

You display an option menu from a button that shows the last item chosen—rather than from the menu title. You can display an option menu by either:

- Clicking the first pointer button on the option menu button.
- Pressing and holding the first pointer button down on the option menu button.

The *periodic* demo provides a dummy "Days of the Week" option menu. Figure 9-4 shows the option menu button and the menu itself. (You can always tell an option menu button by the small rectangle that decorates it.)

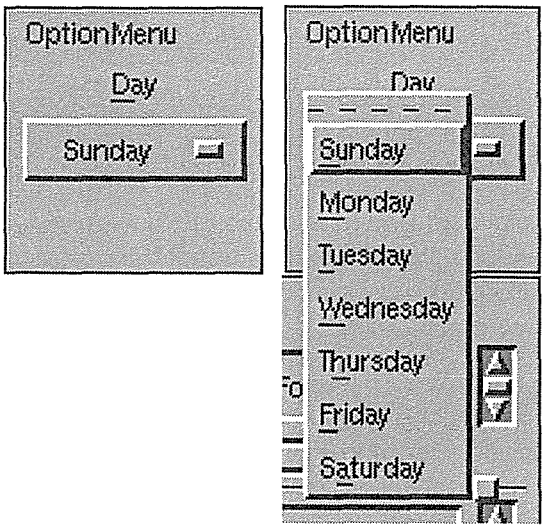

Figure 9-4. Sample option button and option menu

If you display an option menu by the former method, you can select an item by clicking on it with the first pointer button. If you display the menu using the latter method, drag the pointer down the menu and release on the item you want or type the mnemonic abbreviation (the underlined letter).

When you select an item, the menu disappears and the option menu button then displays your selection. To remove an option menu without making a selection, click elsewhere or release the pointer off of the menu, as appropriate.

Tear-off Menus

Pull-down, pop-up, and option menus may also be "tear-off" menus, which you can post in subwindows that remain on the display until you remove them. This feature is very handy if you use a menu frequently. If a menu is not torn off, it disappears after you select an item.

If a menu can be torn off, it will have a "perforated" line as the first item on the menu. Regardless of the menu type, you tear off a menu by clicking or releasing the pointer on this dotted line. For example, to tear off a pull-down menu, you could:

1. Post the menu by clicking the first pointer button. In Figure 9-5, we've again posted *periodic*'s File menu. The perforation below the title indicates that the menu can be torn off.

2. To tear the menu off, click the first pointer button on the perforation. Figure 9-6 shows the torn off File menu.

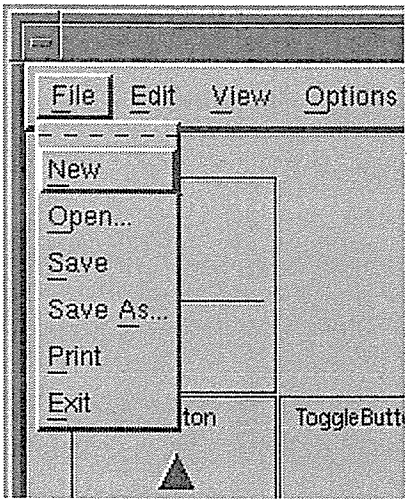

Figure 9-5. Perforation means you can tear off menu

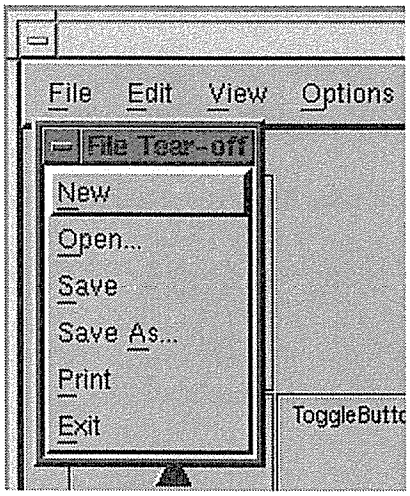

Figure 9-6. Click on the perforation and the tear-off is displayed

To tear off a pop-up menu:

1. Display the pop-up menu. In many cases, you do this by pressing and holding down the third pointer button.
2. Drag the pointer down the menu, highlighting the perforated line.
3. Release the pointer button. The menu is torn off.

Once you tear off a menu, you can work with it much as you would with any application window, with a few limitations. These limitations are borne out by the File tear-off menu's modified *mwm* frame. It has no Minimize or Maximize buttons and no resize handles. The frame does offer a Window Menu button. However, if you display the Window Menu from the

tear-off window, you'll see that it offers only three items: Move, Lower, and Close. Basically, you can move the tear-off window, lower it in the stack, and remove it—and that's all. In Figure 9-7, we've moved the menu out of the way of the *periodic* application window.

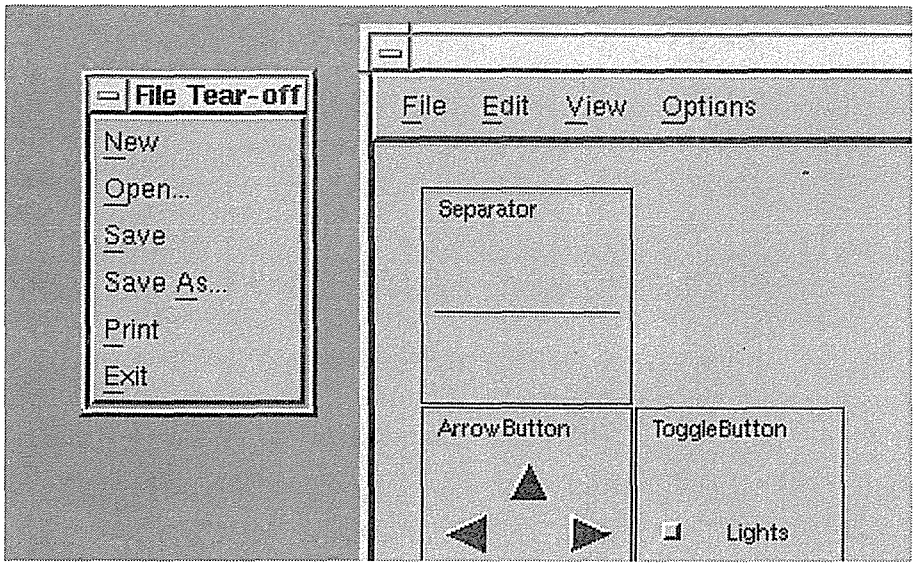

Figure 9-7. Moving a tear-off to a convenient place

There are several ways to remove a tear-off menu. First, direct the input focus to the menu and then perform any one of the following actions:

- Press the key that performs the Cancel function (often Escape).
- Double-click the first pointer button on the menu's own Window Menu button.
- Display the menu's Window Menu and select the Close or type its mnemonic abbreviation, c.
- Use the keyboard accelerator for the Close item, Alt-F4.

Push Buttons

Many Motif applications feature *push buttons* (PushButton widgets). Commonly, a push button has a text label corresponding to some function. You invoke the function by clicking the first pointer button on the push button widget. Figure 9-8 shows the sample push button from the *periodic* demo.

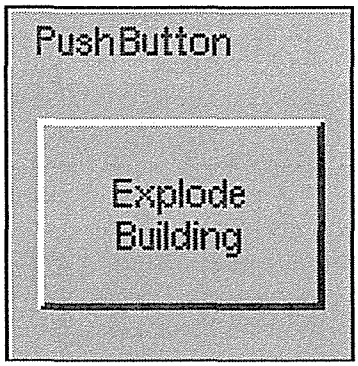

Figure 9-8. Click on a push button to invoke the function

In a real application, clicking on the push button causes some action to occur. Unless James Bond has been working on the *periodic* demo, we can safely assume this button is a dummy! However, you *can* click on it to get an idea what a push button looks like when pressed.

A dialog box always contains one or more push buttons that allow you to respond to the message in the box, but push buttons are also used in other applications. Regardless of the application, in many cases one push button will be highlighted, generally by outlining. If click-to-type focus is in effect, you can push the highlighted push button simply by pressing the Return key on your keyboard. To push another button, you must place the pointer on it and click the first pointer button. (With pointer focus, you need to click on any choice.)

Note that if you press the first pointer button on a push button and then change your mind about invoking the function, there is an escape hatch. Simply move the pointer off the push button before you release the pointer button. (The action of releasing the pointer is actually what invokes the push button.)

An Athena widget set provides a command button with virtually the same functionality as a Motif push button. The most obvious difference is that you must click on an Athena command button to invoke it. The Return key shortcut only works with a Motif push button (click-to-type focus must also be in effect). See "Dialog Boxes and Command Buttons" in Chapter 7, *Graphics Utilities*.

We'll come back to Motif push buttons later, in the discussion of dialog boxes. But before moving on, let's take a quick look at another type of button, called a *drawn button* (DrawnButton widget). From a user's perspective, a drawn button is a push button decorated with a pixmap rather than a text label. You invoke a drawn button just as you do a push button—by clicking the first pointer button on it.* Figure 9-9 shows a drawn button from the *periodic* window. (Note that you can toggle the image on this button on and off using the sample ToggleButton widget in the *periodic* window, which is illustrated in the next section.)

*Push buttons and drawn buttons actually differ from a programming perspective, but a user can expect to interact with them in the same way.

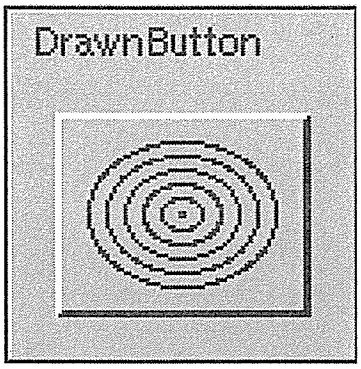

Figure 9-9. A drawn button

The drawn button from the *periodic* demo is simply illustrative—it does nothing. Generally, however, the image on a drawn button will signal its function. If an application uses drawn buttons effectively, they can transcend language barriers, as well as enhance the program's aesthetics. For example, a button decorated with the image of a paint brush might invoke graphics capabilities, a button decorated with the image of a keyboard might open a text editor, etc.

Note that some applications may dynamically change the image on a drawn button to signal some change in the state of the program.

Radio Boxes and Toggle Buttons

A *radio box* is made up of a column of mutually exclusive choices, each represented by a *toggle button* (ToggleButton widget). Figure 9-10 shows a radio box from the *periodic* demo program.

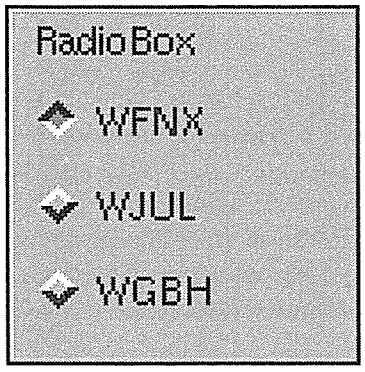

Figure 9-10. A radio box

The column is a single radio box. Several radio boxes may appear side by side in an application. Typically, a radio box contains several diamond-shaped toggle buttons. The radio box

from the *periodic* demo features three buttons, in this case corresponding to—what else—three radio stations. You push a toggle button by placing the pointer on the diamond symbol (or the corresponding text label) and clicking the first pointer button. The toggle button becomes darker (appearing as if it's been pressed). Actually, if you examine the button closely, the highlighting has just switched from the bottom edge to the top edge of the button. In our example, the button next to WFNX is currently selected.

When you first make a selection, the button and the accompanying text label are highlighted by a box. When you make another selection in the same or another column (radio box), the highlighting box appears around that item (and disappears from the previous one).

Toggles in the same radio box are mutually exclusive. If you select one and then select another from the same column, the first one is toggled off. (The button appears to pop up—i.e., the highlighting switches back to the bottom edge of the button; also the highlighting box appears around the latest selection.)

In addition to appearing in radio boxes, toggle buttons may also appear in columns known as *check boxes*. In a check box, toggle buttons *are not* mutually exclusive. You can "check" multiple items in the same column by toggling the corresponding buttons (or labels). The toggle buttons in a check box appear square-shaped to distinguish a check box from a radio box (in which the toggles appear diamond-shaped).

Figure 9-11 shows one such toggle button, the sample from *periodic*, as it appears both off and on. Notice that toggling the button on switches the highlighting from the lower-right corner (it appears to be raised) to the upper-left corner (it appears to have been pushed).

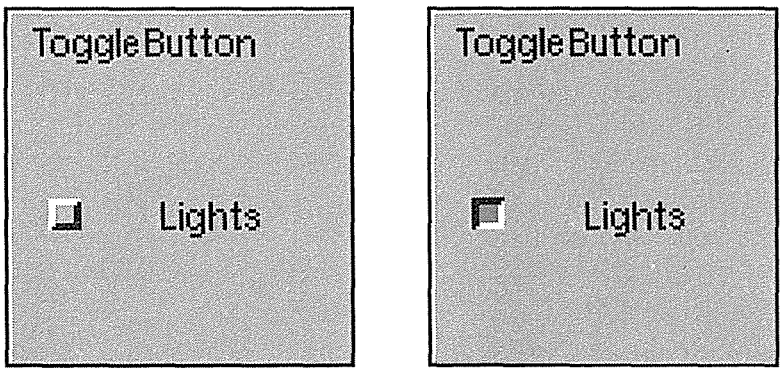

Figure 9-11. Lights off, lights on

Note also that toggle buttons are sometimes labeled with graphic images rather than text. One image represents "on" and another represents "off."

The Motif Scrollbar

Each of the list boxes in the file selection box features both a horizontal and a vertical scrollbar. A vertical scrollbar is commonly used to review text that has scrolled off the top of a window or extends past the bottom. In the case of the Files box in *periodic*'s file selection dialog, the vertical scrollbar is used to scan a list of files too long to fit in the window at one time. A horizontal scrollbar is commonly used to view text or graphics that are too wide to fit in the viewing area. You'll probably encounter vertical scrollbars most often.

Both the Motif and Athena widget sets provide scrollbar widgets. A Motif scrollbar differs in both appearance and operation from an Athena scrollbar, such as the one used by *xterm*. As you know, an Athena scrollbar is simple in design—just a rectangular thumb within a rectangular scroll region. Both parts are flat; the thumb is distinguished from the scroll region only by its (generally) darker color. While a Motif scrollbar has separate parts to invoke different types of scrolling, the Athena scrollbar moves text according to which pointer button you use and how you use it. (See Chapter 5, *The xterm Terminal Emulator*, for instructions on how to use *xterm*'s scrollbar.)

Take another look at the Files box from *periodic*'s file selection box, which is bordered by two scrollbars. A Motif scrollbar is comprised of four parts: two *arrows* (one at either end of the bar), the *scroll region* between the arrows, and the *slider* (analogous to the Athena scrollbar's "thumb"), the raised area that moves within the scroll region. The slider displays the position and amount of text currently showing in the window relative to the amount saved. If text does not extend beyond the window, the slider fills the entire scroll region. In Figure 9-17, the sliders in both scrollbars indicate that text extends beyond the bounds of the window.

Let's consider the pointer commands used to operate a vertical scrollbar. (You'll probably use a vertical scrollbar most often.) To scroll the text forward one window, place the pointer below the slider and click the first button. To scroll the text back one window, place the pointer above the slider and click the first button. In text-based applications, clicking on one of the arrows scrolls the text one line at a time: each click on a down arrow lets you view one more line of text at the bottom of the window; each click on an up arrow lets you view one more line of text at the top of the window.

A horizontal scrollbar lets you view the remaining part of lines that are too wide to fit in a single window. You use the same pointer commands to use a horizontal scrollbar as you do a vertical scrollbar; obviously the orientation of text and directions of movement are different. Clicking to the right of the slider scrolls the text horizontally to the right. Clicking to the left of the slider scrolls the text horizontally to the left. In Figure 9-17, the Files box is displaying filenames only—the earlier parts of the pathnames are not in view. Notice that the horizontal scrollbar's slider is all the way to the right of the scroll region. If you place the pointer to the left of the slider and click the first button, the text is scrolled to the left to reveal the earlier parts of the pathname. In text-based applications, clicking on either arrow of the horizontal scrollbar moves the text one character to the left or right, depending on the direction of the arrow.

Regardless of the orientation of the scrollbar, you can drag the slider by placing the pointer on it and holding down the second pointer button. The text in the window follows the motion of the slider. Release the pointer button when the window displays the text you want.

A final note: the unit scrolled when you click on an arrow varies by application. For instance, scrollbars are sometimes featured on application windows that contain graphic elements rather than text. Obviously, such a window cannot be scrolled by text characters or lines. For example, clicking the pointer on a scrollbar arrow in the *mwm* icon box (described in Chapter 13), scrolls the box the height or width of one icon.

Text Windows

We've already seen several instances of the Athena Text widget, which allows you to enter text in standard X clients like *xedit*, *xman*, *bitmap*, etc. Many Motif applications also provide areas in which you can enter text. A text window may be as small as a single line (many dialog boxes provide one-line text windows) or it may accommodate a file of any length. Regardless of the size of a Motif text window, there are various commands you can use to enter text. This section explains some of the more useful commands.

As you may recall, in most standard X applications, the text cursor is a caret. In Motif text windows, the text cursor is the I-beam symbol. (We've encountered the I-beam cursor before, in a different context: the root window pointer becomes an I-beam when it rests in an *xterm* window.)

Figure 9-12 shows the sample text window from the *periodic* demo. (This is a simple window; in many cases, text windows will also have scrollbars.)

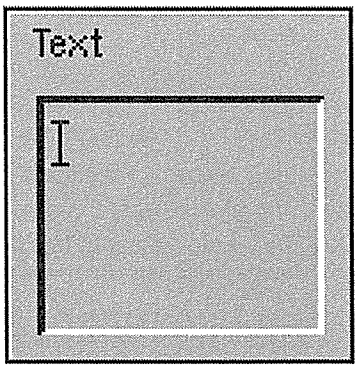

Figure 9-12. Sample text window

The I-beam cursor in the text window marks the point at which text can be inserted. When the window has the input focus, you can begin to enter text simply by typing. Backspace over characters to erase them. To erase multiple characters at once, do the following steps:

1. Select the text by holding the first pointer button and dragging (as you would in any *xterm* window). If the text you want to highlight extends beyond the bounds of the text window, move the pointer outside the window; the window will scroll and additional text will be highlighted. Once you've selected the text you want, release the pointer button.
2. Press the Delete key to erase the selected text.

You can move the I-beam cursor to another insertion point in the text by moving the root window pointer (often represented by an arrow) and clicking the first pointer button. You can also move the cursor within the text using various keyboard commands, summarized in Table 9-1.

Table 9-1. Keyboard Commands to Move the Text Cursor

Keystrokes	Cursor movement
Left arrow	Back one character
Right arrow	Forward one character
Up arrow	Back one line
Down arrow	Forward one line
Control-Left arrow	Back one word
Control-Right arrow	Forward one word
Control-Up arrow	Back one paragraph
Control-Down arrow	Forward one paragraph
PageUp	Back one window
PageDown	Forward one window

The location of the PageUp and PageDown keys may vary per site. In some cases, these keys may be clearly marked. If not, these functions may or may not be assigned to keys and you'll have to perform a little detective work if you want to use them. First, find out if your system administrator has copied a file called *.motifbind* to your home directory. This file maps functions commonly used in Motif applications to convenient keys on your keyboard. If you have a *.motifbind* file, you can use it (along with the *xev* client, described in Chapter 14) to determine the location of keys like PageUp and PageDown. Note, however, that in most cases, you can either fall back on the less powerful Control key combinations—or use scrollbars, if they are provided.

You can copy (or cut) and paste text within a Motif text window or between windows using a few different methods. We'll describe two of them here. See the section "Drag and Drop" for instructions on transferring text using a more graphic interface.

The first method is virtually the same as one of the *xterm* copy and paste methods. We've already explained the selection method under number 1 in the instructions to delete a passage.

1. Select text by pressing and holding down the first pointer button, dragging, and releasing.
2. Move the pointer to the place at which you want to insert the selected text.

3. Click the second pointer button to insert a copy of the text. Press Shift and click the second button to move the text to this position (i.e., the text is cut from the initial selection and copied to this place).

If you've copied the text, so long as it remains highlighted in the first position, it will be pasted when you click the second pointer button. To remove the highlighting, move the pointer outside the highlighted area (but keep it within the same text window) and click the first button again. This action will also move the text cursor. If you want to move the cursor without removing the highlighting, press Control and click the first button in the position you want.

A second copy (or cut) and paste method works in the opposite manner. First you select the destination, then the text to be placed there.

1. Move the pointer to the place you want the text to be inserted and click the first button. (The I-beam moves there.)
2. Select the text you want to copy by pressing the Alt (or Meta) key, holding down the second pointer button, and dragging. Text selected in this manner is underlined rather than highlighted in reverse video. (If you begin by pressing and holding the Shift key in addition, the text will be cut rather than copied.)
3. Release the second pointer button and the selected text appears at the insertion point.

Motif provides many ways to perform most functions. See "Drag and Drop" for a discussion of still another text transfer method.

Dialog Boxes

If you've tried to restart *mwm* from the window manager's Root Menu, you've already encountered a Motif dialog box. When you select the Restart menu item, the dialog box pictured in Figure 9-13 is displayed.

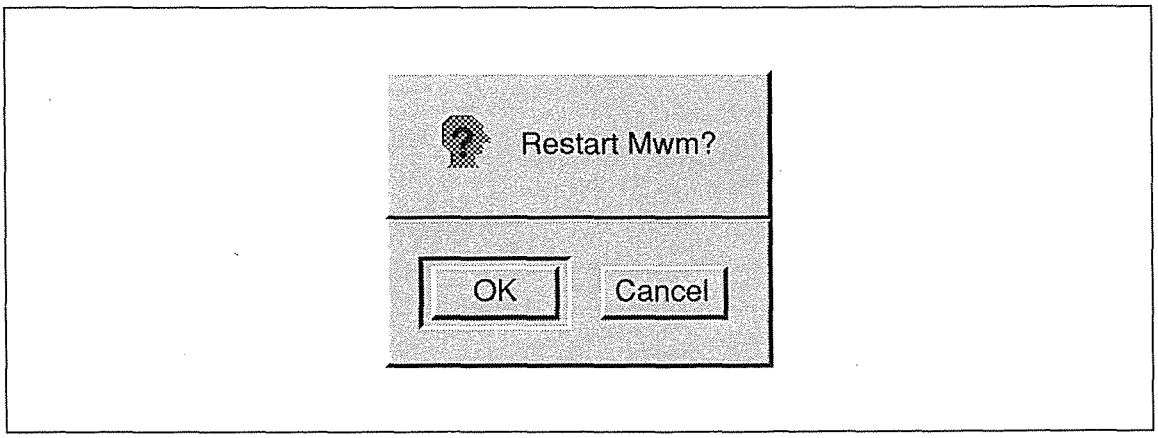

Figure 9-13. Typical Motif dialog box with two push buttons

This sample is a "message" dialog: it displays a message relevant to the application and requires a response from the user. In this case, the dialog box queries whether you really want to `Restart mwm?`.

The *periodic* demo program provides nine sample dialog boxes, which you can display by clicking on one of the buttons in Figure 9-14.

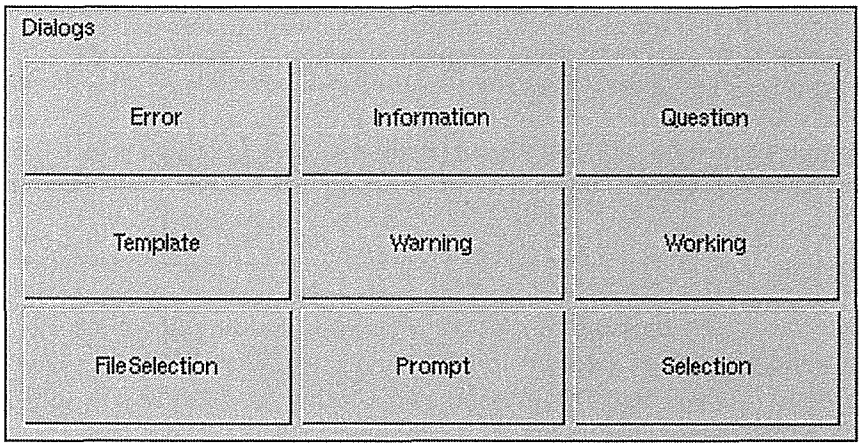

Figure 9-14. Nine push buttons to display periodic's sample dialogs

The first six dialog boxes listed (if you're reading left to right) are all message dialogs, each with a slightly different purpose. The dialog in Figure 9-13 fits into the category of "Question" dialogs. The appropriate response to any message dialog should be obvious. We don't have to consider all six samples here, but browse through them if you like. To display any of the sample dialogs, simply click the first pointer button on the box corresponding to its name.

Regardless of the purpose of a Motif dialog box, it always contains one or more push buttons that allow you to respond to the message. When a dialog is displayed and the default click-to-type focus is in effect, the input focus is usually switched to the dialog window. Until you respond to the dialog box, the application cannot continue. Once you respond to the dialog, the focus should switch back to the main application window.

Whether the dialog box contains one push button or multiple buttons, one button is always highlighted, generally by outlining. If click-to-type focus is in effect, you can activate the highlighted push button simply by pressing the Return key on your keyboard. To push another button, you must place the pointer on it and click the first pointer button. (With pointer focus, you need to click on any choice.)

A response might be a simple acknowledgment that you've seen the message: some dialogs feature only one button that reads OK. For instance, say you invoke a text editor on a particular file and that file does not exist. The program may display a dialog with a message similar to the following:

```
Couldn't open /home/val/vacation.
```

with an OK button. When a dialog has only one button, the button is always highlighted. Pressing Return or clicking the first pointer button on the OK button informs the client that you've seen the message and removes the dialog window.

Some responses request an action, such as proceeding with a previously invoked process, cancelling the process, or even exiting the program. The dialog box in Figure 9-13 contains two push buttons labeled OK and Cancel. Pushing the OK button tells *mwm* to proceed with the restart process. The Cancel button gives you a chance to avert the restart process in case you invoked the command by mistake or have changed your mind. Since Cancel is highlighted, you can push it either by pressing Return or by using the pointer.

Whatever the message or potential responses, you react to a dialog box either by pressing Return (to push the highlighted push button) or by placing the pointer on one of the push buttons and clicking the first pointer button. Action will be taken if requested and the dialog box will be removed.

The Athena widget set provides comparable widgets to the Motif dialog box and push button. An Athena dialog box provides virtually the same functionality as a Motif dialog. The most obvious difference is that, in an Athena dialog, you must click on a command button to invoke it. The Return key shortcut only works with a Motif push button (click-to-type focus must also be in effect). See "Dialog Boxes and Command Buttons" in Chapter 7, *Graphics Utilities*, for more information about Athena dialogs.

In the next few sections, we'll consider some more specialized Motif dialog boxes.

Prompt Dialog

Typically, a prompt dialog box asks the user to supply some small item of information, such as a filename. Figure 9-15 shows the sample prompt dialog provided by *periodic*.

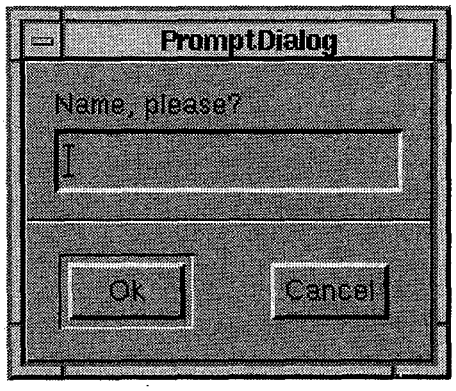

Figure 9-15. A prompt dialog box

In this mock prompt dialog, you enter the "name" you want in the one-line text window. To confirm your entry, you can either press Return or click on OK push button. Click on Cancel to pop down the dialog box without specifying a name.

Selection Dialog

A *selection box* (SelectionBox widget) is a composite dialog that provides a list of items from which you can make a selection. Figure 9-16 shows *periodic*'s sample selection box.

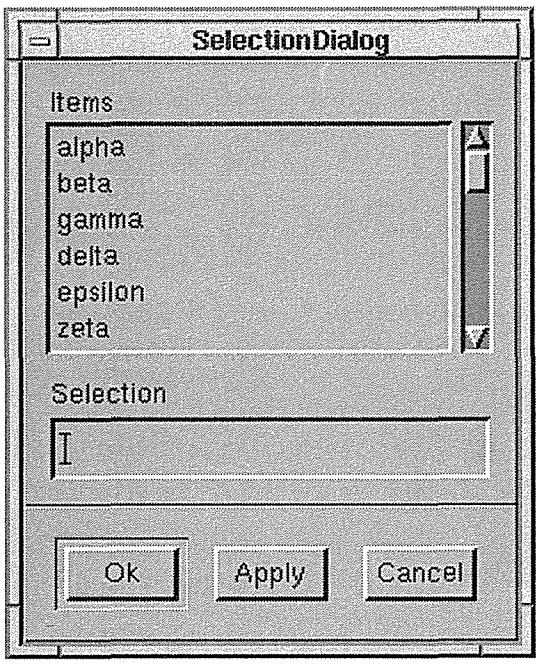

Figure 9-16. A selection dialog box

Using a selection box is fairly simple. (Things become slightly more complicated with a file selection box, described in the next section.) A selection box is generally composed of a *list box* (in this case labeled Items), a one-line text window labeled Selection, and a few push buttons.

Notice that the list box has a vertical scrollbar, which allows you to view text that is currently outside the box. A list box and its accompanying scrollbar(s) form what is known as a ScrolledWindow. (This composite widget is contained in the even more complicated SelectionBox widget!) The Motif ScrolledWindow is comparable to the Athena Viewport widget, discussed in the section "Browsing Reference Pages: xman" in Chapter 8.

You want to place the name of your selection in the one-line Selection window. To do this, you can:

1. Place the pointer on the item you want in the Items list box. (You can use the scrollbar to view additional possibilities.)
2. Click the first pointer button. The item is highlighted and appears in the Selection window.

(Of course, since the Selection window is a text window, you might instead choose to type the name yourself.) Then you can confirm your selection by pressing Return or clicking on the appropriate push button (in this case, OK); or pop down the box without making a selection by clicking on the Cancel push button.

File Selection Dialog

Several Motif applications feature a rather complicated type of dialog called a *file selection box* (FileSelectionBox widget), which allows you to browse a directory structure and select a file. A file selection box is similar to a selection box, but is a bit more complicated.

Using a file selection box is not exactly difficult, but it's not particularly obvious either. Let's consider the file selection box that is displayed when you click on *periodic*'s FileSelectionDialog button. The box appears in Figure 9-17.

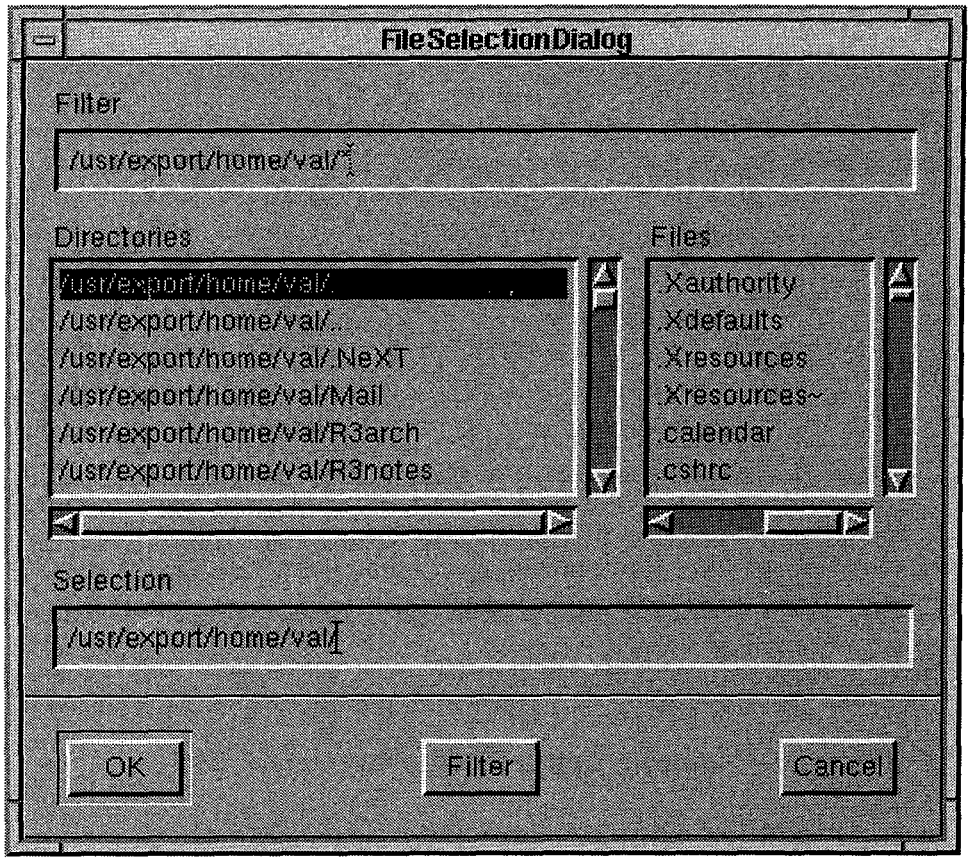

Figure 9-17. A file selection dialog box

As in the selection box, the file selection features a window labeled Selection near the bottom of the box. You want to place the name of the file to select in this window. Initially this window contains an incomplete pathname—a directory is specified but no file. You can specify a file in a variety of ways.

Notice the two areas labeled Directories and Files. These are list boxes that are contained within the larger window. The Directories box lists the directories from which you can choose a file; the first directory is usually highlighted. The Files box lists the files within the highlighted directory. (In your version of *periodic*, this box may be labeled List, but this is anomolous; virtually all applications have a Files box instead.)

Notice that the list boxes are bordered by both horizontal and vertical scrollbars, which allow you to view text that is currently outside the box. These list boxes are ScrolledWindow widgets contained in the more complicated FileSelectionBox widget.

The file selection box allows you to select a file from any directory on the system, using various procedures. You can select a file from the list currently in the Files box; you can list the files in another directory currently displayed in the the Directories box and select one of those files; or you can list the contents of an entirely different directory and select a file from that directory.

Selecting a File from the Files Box

To select a file currently in the Files box:

1. Place the pointer on the filename.
2. Click the first pointer button. The filename is highlighted by a dark bar; the letters appear in reverse video.

   The Selection window will be updated to reflect the filename; and the push button to confirm the selection (OK in many applications) will be highlighted, indicating that you can select the file by pressing Return.
3. Select the filename either by pressing Return or by placing the pointer on the OK or other appropriate push button and clicking the first pointer button.

When you select a file, the file selection box disappears.

Choosing a File from Another Directory in the Directories Box

To list the files in another directory in the Directories box and select one of those files:

1. Place the pointer on the directory name and click the first button. The directory name is highlighted. Notice that the box labeled Filter is updated to reflect the new pathname and the Filter push button at the bottom of the box is highlighted for selection.
2. Then, to display the contents of the highlighted directory in the Files box either:
   - Press Return; or
   - Click on the Filter push button.

3. To select a file from the updated Files box, follow the steps outlined previously in "Selecting a File from the Files Box."

Choosing a File from Another Directory on the System

You can specify an alternative directory from which a file can be selected by changing the filter, that is, the path in the Filter window (near the top of the file selection box). Initially the Filter window reflects the current working directory. In Figure 9-17, the filter is */usr/export/home/val/** and the Directories box lists several directories:

```
/usr/export/home/val/.          \"the current directory
/usr/export/home/val/..         \"previous directory in the tree
/usr/export/home/val/.NeXT      \"subdirectories
/usr/export/home/val/Mail       \"
/usr/export/home/val/R3arch     \"
/usr/export/home/val/R3notes    \"
                .
                .
                .
```

The vertical scrollbar indicates that there are several more directories in the list (which you can browse using the Motif scrollbar commands). To specify another filter, place the pointer within the Filter window and double click the first pointer button. The window becomes highlighted with a black bar (the text is visible in reverse video); now whatever you type will replace the current text.

When you type a pathname and hit Return (or click on the Filter push button at the bottom of the file selection box), the Directories box will be updated to reflect the filter you've specified. For example, if you enter the following pathname in the Filter window:

```
/usr/export/home/paula/*
```

and hit Return or click on the Filter push button, the Directories box will be updated to reflect the directory */usr/export/home/paula*, its subdirectories, and the directory above it in the tree. The first directory in the Directories box, */usr/export/home/paula/.*, will be highlighted and the files in that directory will appear in the Files box.

You can then choose any of the files in the Files box using the steps outlined previously in "Selecting a File from the Files Box."

Command Box

A *command box* (Command widget) is a composite widget that operates something like a selection box—only in the reverse. You enter a command (presumably to be invoked) in a one-line text field at the bottom of the box; that command is then echoed in a larger text window above the command field. (The larger text window maintains a history of the commands you enter. You cannot edit this history.) Figure 9-18 illustrates *periodic*'s sample command box. Although *periodic* uses a command box in the main application window, they are often used in dialog boxes. (The Command widget could legitimately appear as a tenth sample dialog; thus, we've included it here.)

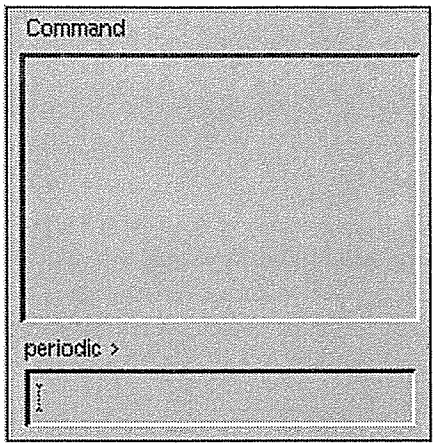

Figure 9-18. A command box

The command entry window is at the bottom of the box; it is generally introduced by a prompt, in this case:

```
periodic >
```

Direct the input focus to the small text window, type the name of a command, and press Return. The command window is cleared and the command name appears in the history window above, as in Figure 9-19. (In a functioning application, the command would also be executed.)

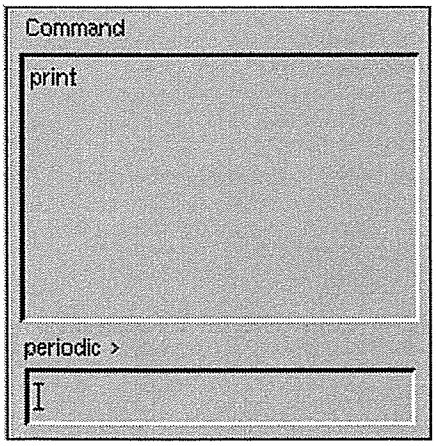

Figure 9-19. Command entered in small text window appears in history window

When you enter subsequent commands, they appear on subsequent lines in the history window. Figure 9-20 illustrates entering a second command.

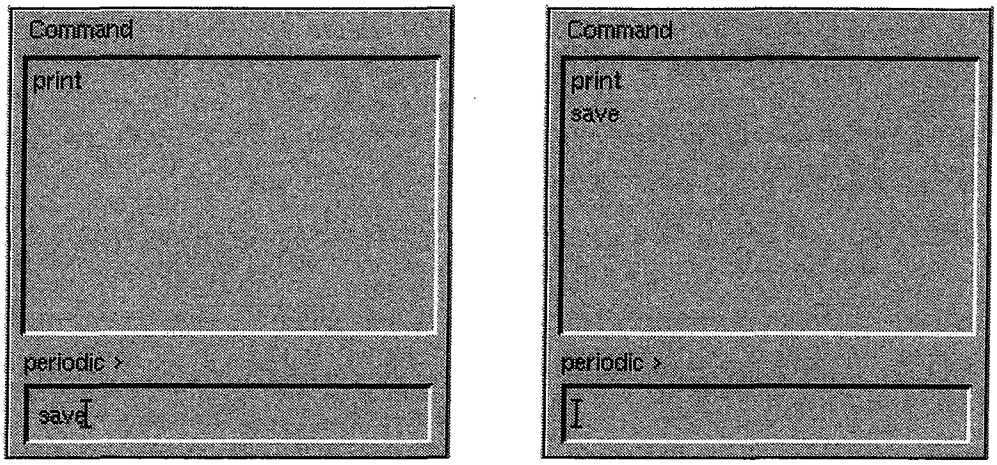

Figure 9-20. Entering another command: before and after

If you click on any item in the history window, it will be highlighted and the text will appear in the smaller command window. This provides a way to repeat commands without retyping. You can then edit the command in the small command window, if you want. Remember, however, that you cannot edit text in the history window.

Scale

A *scale* (Scale widget) displays a numeric value within a range of values. A scale consists of a narrow, rectangular trough that contains a slider. The slider's position marks the current value within the range. Typically, the slider is bordered by labels that indicate the unit of measure (which remains static) and the current value (which is updated dynamically). Figure 9-21 shows the scale provided by the *periodic* demo, which represents a range of radio bandwidth in megahertz.

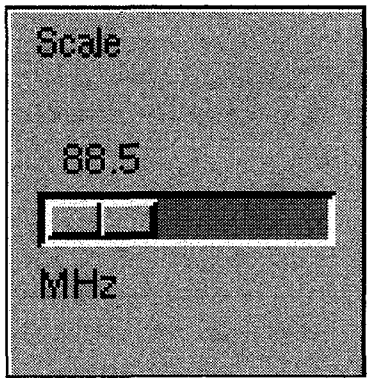

Figure 9-21. Scale widget

In the sample, the unit of measure is mHZ (megahertz); the current value is 88.5. In some implementations, the user can adjust the value by moving the slider. In other cases, the scale simply registers changes to the numeric value (presumably tracked by the program)—the user cannot modify it. The *periodic* sample allows you to change the value. You can move the slider (and change the value) in the following ways:

- Clicking the first pointer button within the trough on either side of the slider increases or decreases the value by one unit.
- Clicking the second pointer button anywhere within the trough causes the slider to move to that position and the value to be changed accordingly.
- Holding the first pointer button down on the slider allows you to drag the slider within the trough. (You release the pointer button when you've positioned the slider where you want it.)

Try dragging the slider by holding down the first pointer button. As you move the slider, notice that the value changes. In Figure 9-22 we've dragged the slider to the right and the value has increased to 100.1.

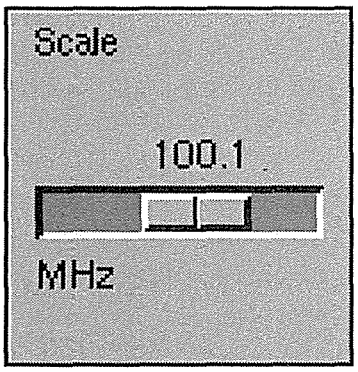

Figure 9-22. When you move the slider, the value changes

In our example, the scale is horizontal, but a scale may also be vertical. Some scales have "tick marks"—labels that indicate values within the range. In some cases, a scale will also have arrow buttons at either end (similar to the scrollbar). Clicking the first pointer button on either arrow increases or decreases the value by one unit.

Drag and Drop

Motif 1.2 provides an additional way to transfer information within an application and between applications: "drag and drop." Basically, you select an element (text, graphics, etc.) from one place, drag a copy of it (or sometimes the original) to another place, and drop it in. In many applications, drag and drop simply transfers information, but in some cases, you might drag an image and drop it to invoke an action.

When used to transfer information, drag and drop presumes that the two windows interpret data in the same format. You can't drag a graphic image and drop it in a text window, for instance. Keep in mind that drag and drop greatly depends on application-specific factors. Some elements in an application may be available for selection (and transfer), but others won't. Without considering application-specific enhancements, you can assume that the following data can be dragged:

- Text within text windows
- Labels (on push buttons, toggle buttons, etc.)
- One or more items within a list

The only default "drop site" for this data is a Motif text field.

As of the publication of this guide, commercial applications had not yet taken advantage of drag and drop functionality, so our discussion is necessarily limited. This section gives an overview and some examples from the *periodic* program. These guidelines should help when you begin using commercial applications that support drag and drop, but also consult the individual program documentation.

In our discussion of the Text widget, we considered two methods of transferring text. Drag and drop represents a third method. To drag text from one place to another:

1. Select (highlight) the text using the first pointer button (as described in the "Text Windows" section).
2. Place the pointer on the selected text and press and hold the second pointer button. An icon representing the information being dragged is displayed. (If you additionally press the Shift key, the text will be moved, rather than copied. If you press Control or no key, the text will be copied.)
3. Drag the icon to the position you want to place the text (within the same or another text window). Note that the text cursor in that window should be at the proper insertion point.
4. Release the second pointer button. (If you're holding any modifier keys, continue to hold them until the button is released.) If you've selected a valid drop site, the icon disappears and the text is inserted. If the drop site is not valid, the icon appears to spring back to its source.

The so-called "drag icon" varies depending on the application and the type of information being dragged. It may also change dynamically to indicate whether the pointer is over a valid drop site.

You can also drag items from a list and drop them in a Text widget, using virtually the same steps. If you want to drag a single item, there's no need to select it first; just follow steps two through four above. To drag multiple list items:

1. Select the items you want. (Press and hold the first pointer button and drag to select multiple items in a row.)
2. Complete the drag and drop using steps two through four from the procedure to transfer text.

Finally, you can drag a label and drop it in a Text widget. There's no need to select it first. Simply follow steps two through four from the procedure to transfer text.

To cancel a drag operation, while you continue to hold the second pointer button, press the key that invokes the Cancel function (often the Escape key). The drag icon appears to spring back to its source.

As we've said, application developers will undoubtedly implement drag and drop in additional ways. Motif 1.2 includes a program called *DNDDemo* that allows you to drag one of six colors into a rectangle. If you have a color monitor, playing with this demo might give you a better idea of the possibilities of drag and drop.

Once you run the program, you must first draw the rectangle within a white space in the application window. To do so, press and hold the first pointer button, drag, and release (much as you would draw a rectangle using the standard *bitmap* client). Initially, the rectangle is black. To color it, move the pointer into one of six colored squares arranged along the bottom of the window. Then press and hold the second pointer button to drag the color. The drag icon is an artist's palette in the chosen color.

When you drag the icon over an invalid drop source (anywhere except the rectangle), a red international negation symbol is superimposed over the palette, indicating that you cannot make the transfer. When the icon is over the rectangle, the palette appears normal. Release the pointer over the rectangle and the shape is redrawn in the selected color. You can change the color as many times as you like. Of course, this is only mildly diverting, but it will give you an idea how drag and drop might be implemented in the future—in more graphically-oriented applications.

10

Command-line Options

This chapter describes command-line options that are common to most clients. Some arguments to command-line options can also be specified as the values of resource variables, described in Chapter 11, Setting Resources. *For example, the format of a geometry string or a color specification is the same whether it is specified as an argument to an option or as the value of a resource definition.*

In This Chapter:

10 Command-line Options

As explained in Chapter 3, *Working in the X Environment*, X allows the user to specify numerous (very numerous!) command-line options when starting most clients. The command-line options for each client are detailed on the reference pages in Part Three of this guide.

As a general rule, all options can be shortened to the shortest unique abbreviation. For example, `-display` can be shortened to `-d` if there is no other option beginning with "d." (Note that while this is true for all the standard MIT clients, it may not be true of any random client taken off the net.)

In addition to certain client-specific options, all applications built with the X Toolkit (or a toolkit based on the Xt Intrinsics, such as the Motif Toolkit) accept certain standard options, which are listed in Table 10-1. (Some non-Toolkit applications may also recognize these options.) The first column contains the name of the option, the second the name of the resource to which it corresponds (see Chapter 11, *Setting Resources*), and the third a brief description of what the option does.

This chapter discusses some of the more commonly used Toolkit options and demonstrates how to use them. (For the syntax of the other Toolkit options, see the *X* reference page in Part Three of this guide.)

Table 10-1. Standard Options

Option	Resource	Description
`-bg`	`background`	Background color of window.
`-background`	`background`	Background color of window.
`-bd`	`borderColor`	Color of window border.
`-bordercolor`	`borderColor`	Color of window border.
`-bw`	`borderWidth`	Border width of window in pixels.
`-borderwidth`	`borderWidth`	Border width of window in pixels.
`-display`	`display`	Display on which client is run.
`-fn`	`font`	Font for text display.
`-font`	`font`	Font for text display.

Table 10-1. Standard Options (continued)

Option	Resource	Description
`-fg`	`foreground`	Foreground (drawing or text) color of window.
`-foreground`	`foreground`	Foreground (drawing or text) color of window.
`-geometry`	`geometry`	Geometry string for window size and placement.
`-iconic`		Start the application in iconified form.
`-name`	`name`	Specify a name for the application being run.
`-rv`	`reverseVideo`	Reverse foreground and background colors.
`-reverse`	`reverseVideo`	Reverse foreground and background colors.
`+rv`	`reverseVideo`	Don't reverse foreground and background.
`-selectionTimeout`	`selectionTimeout`	Timeout in milliseconds within which two communicating applications must respond to one another for a selection request.
`-synchronous`	`synchronous`	Enable synchronous debug mode.
`+synchronous`	`synchronous`	Disable synchronous debug mode.
`-title`	`title`	Specify a window title (e.g., to be displayed in a titlebar).
`-xnllanguage`	`xnlLanguage`	The language, territory, and codeset for National Language Support; this information helps resolve resource and other filenames.
`-xrm`	value of next arg	Next argument is a quoted string containing a resource manager specification, as described in Chapter 11, *Setting Resources*.

Though all Toolkit options are preceded by a minus sign, client-specific options may or may not require it. See the reference page for each client in Part Three of this guide for the syntax of all options.

Display and Geometry

Perhaps the most useful of the Toolkit options are `-display` and `-geometry`, which allow you to specify the display on which a client window should appear, and the size and position of that window, respectively. See Chapter 3, *Working in the X Environment*, for detailed instructions on using these options. In the remainder of this chapter we'll discuss some of the other useful Toolkit options.

Window Title and Application Name

You can specify the title of a window (as it appears in the titlebar) and the name of the program (as known to the server) using the `-title` and `-name` options, respectively.

The `-title` option allows you to specify a text string as the title of the application's window. If your application has a titlebar or if the window manager you are using puts titlebars on windows, this string will appear in the titlebar. (Note that most applications use the name of the program as the default title.)

Window titles can be useful in distinguishing multiple instances of the same application. For example, say you're running three *xterm* windows, each on a different system in the network. You can give each of the windows a title that matches the system on which the client is running:

```
% xterm -title jersey -geometry +0+0 &
% rsh manhattan xterm -title manhattan -display jersey:0.0 -geometry -0+0 &
% rsh bronx xterm -title bronx -display jersey:0.0 -geometry -0-0 &
```

In this case, the user is working on a workstation named *jersey*. She is running an *xterm* on the local machine and giving it a title to match (`-title jersey`). She is also running *xterm* windows on the remote systems *manhattan* and *bronx*, displaying the windows on *jersey* (using `-display`), and titling each window to match its system. (The `-geometry` option allows her to provide convenient placement for all three windows. See Chapter 3, *Working in the X Environment*, for a complete discussion of the `-display` and `-geometry` options.) The resulting three windows appear in the display in Figure 10-1.

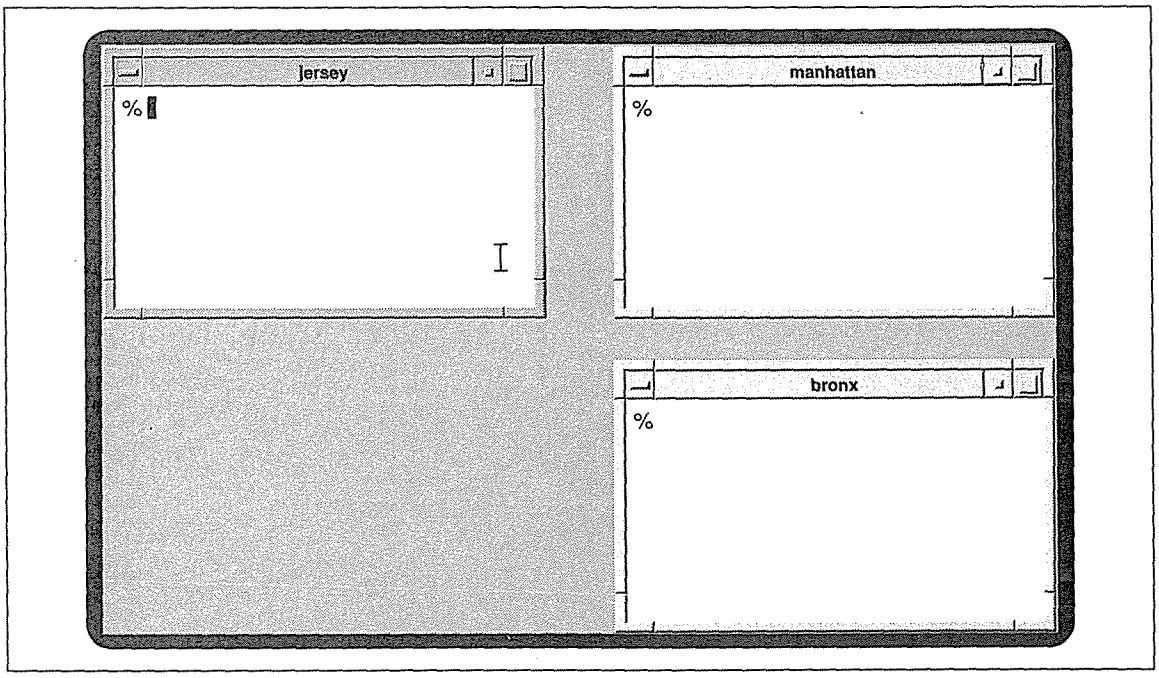

Figure 10-1. Window titles showing client's host system

Specifying the machine name as the title string is just one use of `-title`. You might choose to title a window based on any number of factors, perhaps even its intended function. For instance, you might have windows titled *editing*, *mail*, *sales*, *book project*, etc. If you want to specify a title that is composed of multiple words, enclose the title in quotation marks:

```
% xterm -title "X Window System User's Guide, Motif Edition" &
```

The `-name` option actually changes the name by which the server identifies the program. If a name string is defined for an application, that string will appear as the application name in its icon. More significantly, using `-name` to change the name of the application itself affects the way the resource definitions are applied. By renaming one instance of a client, you can specify resources that apply only to that renamed version. Because the new name can be used in resource definitions, it should be limited to a single word. The `-name` option is discussed further in Chapter 11, *Setting Resources*.

If you display information about a currently running window using the *xwininfo* client (without options), title strings will appear in parentheses after the associated window ID numbers. If there is no title string but there is a name string, the name string will be displayed. If you use the `-tree` option (to list information about the window tree), both title and name strings are returned.

You can also use the *xwininfo* client to request information about a particular window by title, or name, if no title string is defined, using that application's own `-name` option. See the *xwininfo* reference page in Part Three of this guide and the section "Window and Display Information Clients" in Chapter 8, *Other Clients*, to learn more about this client.

Displaying the Current Directory in an xterm Titlebar

Without customization, an *xterm* window's titlebar will display simply the program name ("xterm"). Of course, you can specify alternative text to be displayed in the titlebar using the `-title` option.

You can use a somewhat fancier trick—employing a special escape sequence—to get the titlebar to display the current working directory. If you're running the UNIX C shell, you can do this by writing an alias for *cd*(1):

```
alias cd 'chdir \!*;echo -n "Escape]2;$cwdControl-G"'
```

With this alias, each time you change directory (with *cd*), an escape sequence is echoed to the *xterm* shell. The escape sequence tells the *xterm* to update the titlebar text to reflect the *cwd* environment variable (which contains the current working directory). (See Appendix E, *xterm Control Sequences*, for a complete list of valid sequences.)

In the example above, the escape sequence is represented by the literal keys you type, which are:

```
Escape ] 2 ; $cwd Control-G
```

(continued on next page)

Displaying the Current Directory in an xterm Titlebar (continued)

Be aware, however, that when you type these keys as specified, the command line will not look exactly like this. Certain keys, such as Escape, and key combinations, such as Control-G, are represented by other symbols on the command line. When you type the previous key sequence (without spaces), the command line will actually look like this:

```
alias cd 'chdir \!*;echo -n "^[]2;$cwd^G"'
```

Pressing the Escape key generates the ^[symbol; typing the Control-G key combination generates ^G.

You can specify this alias on the command line or add it to your *.cshrc* file. We recommend entering it in your *.cshrc* file. If you enter it in the file, you'll probably need to preface the special keys Escape and Control-G with other keys to get them to appear. If you're using the *vi* text editor, type Control-V before a special key or key combination. If you're using *emacs*, type Control-Q first. If you're using another text editor, see your documentation or your system administrator for details.

If you're using another UNIX shell or are working in an entirely different environment, consult your system administrator for the proper way to supply the escape sequence to the *xterm* window.

Starting a Client Window as an Icon

The `-iconic` command-line option starts the client window in iconified form. To start an *xterm* window as an icon, type:

```
% xterm -iconic &
```

This can be especially useful for starting the login *xterm* window. As described in Chapter 3, *Working in the X Environment*, terminating the login *xterm* window kills the X server and all other clients that are running. It's always possible to terminate a window inadvertently by selecting the wrong menu option or typing the wrong key sequence. If your login *xterm* window is automatically iconified at startup, you are far less likely to terminate the window inadvertently and end your X session.

Normally, *mwm* handles icon placement, so you shouldn't have to worry about it. By default, icons are displayed in the bottom left corner of the root window. *mwm* can also be set up to place icons in another location, to allow you to place them interactively using the pointer, or to organize icons within an *icon box*. Chapter 13, *Customizing mwm*, describes the specifications necessary to set up an icon box. See the *mwm* reference page in Part Three of this guide for additional information on icon placement.

Specifying Fonts on the Command Line

Many clients allow you to specify the font to be used when displaying text in the window. (These are known as *screen fonts* and are not to be confused with *printer fonts*.) For clients written with the X Toolkit, the option to set the display font is `-fn`. For example, the command line:

```
% xterm -fn fontname &
```

creates an *xterm* window in which text will be displayed with the font named *fontname*.

Chapter 6, *Font Specification*, describes the available screen fonts and font naming conventions.

Reverse Video

There are three options to control whether the application will display in reverse video—that is, with the foreground and background colors reversed. The `-rv` or `-reverse` option is used to request reverse video.

The `+rv` option is used to override any reverse video request that might be specified in a resource file. (See Chapter 11, *Setting Resources*.) This is important, because not all clients handle reverse video correctly, and even those that do usually do so only on black and white displays.

Border Width

Many clients accept a `-bw` option that is intended to specify the width of the window border in pixels. However, if you're using the *mwm* window manager, this customization is generally useless because the *mwm* frame effectively replaces most window borders.

As an alternative, you *can* change the width of the frame by specifying resources for *mwm* in a *.Xresources* or *.Xdefaults* file in your home directory. For more information, see Chapter 11, *Setting Resources*, and the `frameBorderWidth` and `resizeBorderWidth` resources on the *mwm* reference page in Part Three of this guide.

Specifying Color

Many clients accept standard options that allow you to specify the color of the window background, foreground (the color in which text or graphic elements will be displayed), and border. These options generally have the form:

-bg *color* Sets the background color.

-fg *color* Sets the foreground color.

-bd *color* Sets the border color.

By default, the background of an application window is usually white and the foreground black, even on color workstations. The `-bg` and `-fg` options allow you to specify alternatives.

Many clients accept the `-bd` option that is intended to specify the color of the window border. However, as in the case of the `-bw` (border width) option, if you're using the *mwm* window manager, this customization is generally useless: the *mwm* frame effectively replaces most window borders. As an alternative, you can change the color of the frame by specifying resources for *mwm* in a *.Xresources* or *.Xdefaults* file in your home directory. For more information, see Chapter 11, *Setting Resources*, and the *mwm* reference page in Part Three of this guide.

You can name another *color* on the command line in a variety of ways, which are described in Chapter 12, *Specifying Color*. Some color specifications are simply names (blue, green, hot pink); others are symbolic "names"—actually numeric values that signify a particular shade. (As you're probably guessing, this can get complicated.) For now, suffice it to say that you can keep color specification as simple as you want—and most color names you can think of are probably valid.

Let's consider the syntax of a command line specifying an *xterm* to be displayed in two colors:

```
% xterm -bg lightblue -fg darkslategray &
```

This command creates an *xterm* window with a background of light blue and foreground of dark slate gray.

At the command line, you should either type a color name as a single word (for example, `darkslategray`) or enclose the separate words in quotes, as in the command line:

```
% xterm -bg "light blue" -fg "dark slate gray" &
```

As we'll see in Chapter 12, if you specify colors using these "standard" names X allows for a range of spelling, spacing, and capitalization.

Some clients allow additional options to specify color for other elements, such as the cursor, highlighting, and so on. See the appropriate client reference pages in Part Three of this guide for details.

See Chapter 12, *Specifying Color*, for more about the standard color names and customized color values.

11

Setting Resources

This chapter describes how to set resource variables that determine application features such as color, geometry, fonts, and so on. It describes the syntax of resource definition files such as .Xresources, as well as the operation of xrdb, *a client that can be used to change resource definitions dynamically, and make resources available to clients running on other machines.*

In This Chapter:

11
Setting Resources

Virtually all X clients are customizable. You can specify how a client looks on the screen—its size and placement, its border and background color or pattern, whether the window has a scrollbar, and so on. Some applications even allow you to redefine the keystrokes or pointer actions used to control the application.

Traditional UNIX applications rely on command-line options to allow users to customize the way they work. As we've already discussed in Chapter 10, *Command-line Options*, X applications support command-line options too, but often not for all features. Also, there can be so many customizable features in an application that entering a command line to set them all would be completely impractical. (Imagine the aggravation of misspelling an option in a command that was three lines long!)

X offers an alternative to customizing an application on the command line. Almost every feature of a program can be controlled by a variable called a *resource*; you can change the behavior or appearance of a program by changing the *value* associated with a resource variable. (All of the standard X Toolkit *Command-line Options* described in Chapter 10 have corresponding resource variable names. See Table 9-1 for more information.)

Resource variables may be Boolean (such as `scrollBar: True`) or take a numeric or string value (`borderWidth: 2` or `foreground: blue`). What's more, in applications written with the X Toolkit (or an Xt-based toolkit such as the Motif toolkit), resources may be associated with separate *objects* (or "widgets") within an application. There is a syntax that allows for separate control over both a *class* of objects in the application and an individual *instance* of an object. This is illustrated by these resource specifications for a hypothetical application called *xclient*:

```
xclient*Buttons.foreground:        blue
xclient*help.foreground:           red
```

The first resource specification makes the foreground color of all buttons in the *xclient* application (in the class `Buttons`) blue; the second resource specification makes the foreground color of the `help` button in this application (an instance of the class `Buttons`) red. Resource settings can be simpler than this. If you want to set very simple resources, read the next section, "Resource Naming Syntax." You can delve more deeply into a client's widget hierarchy to set more complicated and precise resources. The *editres* client (described later in this chapter) helps you determine a client's hierarchy and set the resources you want.

The values of resources can be set as application defaults using a number of different mechanisms, including resource files in your home directory and a program called *xrdb* (X resource database manager). As we'll see, the *xrdb* program stores resources directly in the server, making them available to all clients, regardless of the machine the clients are running on.

Placing resources in files allows you to set many resources at once, without the restrictions encountered when using command-line options. In addition to a primary resource file (often called *.Xdefaults*, *.Xresources*, *xrdb*) in your home directory, which determines defaults for the clients you yourself run, the system administrator can create system-wide resource files to set defaults for all instances of the application run on this machine. It is also possible to create resource files to set some resources only for the local machine, some for all machines in a network, and some for one or more specific machines.

The various resource files are automatically read in and processed in a certain order within an application by a set of routines called the *resource manager*. The syntax for resource specifications and the rules of precedence by which the resource manager processes them are intended to give you the maximum flexibility in setting resources with the minimum amount of text. You can specify a resource that controls only one feature of a single application, such as the red `help` button in the hypothetical *xclient* settings above. You can also specify a resource that controls one feature of multiple objects within multiple applications with a single line.

As of Release 5, the resource manager also allows you to specify different resources for color and monochrome screens. In addition, you can invoke predefined color defaults for an application by using the new `customization` resource variable.

It is important to note that command-line options normally take precedence over any prior resource settings; so you can set up the files to control the way you *normally* want your application to work and then use command-line options to specify changes you need for only one or two instances of the application.

In this chapter, we'll first look at the syntax of resource specifications. Then we'll consider some methods of setting resources, primarily some special command-line options and the *xrdb* program. Finally, we'll take a brief look at other sources of resource definition, additional files that can be created or edited to set application resources.

Resource Naming Syntax

The basic syntax of a resource definition file is fairly simple. Each client recognizes certain resource variables that can be assigned a value. The variables for each client are documented on its reference page in Part Three of this guide.

Most of the common clients are written to use the X Toolkit. As described in Chapter 1, *An Introduction to the X Window System*, toolkits are a mechanism for simplifying the design and coding of applications and making them operate in a consistent way. Toolkits provide a standard set of objects, or widgets, such as menus, command buttons, dialog boxes,

scrollbars, and so on. As we'll see, the naming syntax for certain resources parallels the object hierarchy that is built into X Toolkit programs.*

The most basic line you can have in a resource definition file consists of the name of a client, followed by a period or an asterisk, and the name of a variable. A colon and whitespace separate the client and variable names from the actual value of the resource variable. The following line specifies that all instances of the *xterm* application have a scrollbar:

```
xterm*scrollBar:    True
```

If the name of the client is omitted, the variable applies to all instances of all clients (in this case, all clients that can have a scrollbar). If the same variable is specified as a global variable and a client-specific variable, the value of the client-specific variable takes precedence for that client. Note, however, that if the name of the client is omitted, the line should generally begin with an asterisk.

Be sure not to inadvertently omit the colon at the end of a resource specification. This is an easy mistake to make and the resource manager provides no error messages. If there is an error in a resource specification (including a syntax error such as the omission of the colon or a misspelling), the specification is ignored. The value you set will simply not take effect. To include a comment in a resource file or comment out one of the resource specifications, begin the line in question with an exclamation point (!). If the last character on a line is a backslash (\), the resource definition on that line is assumed to continue on the next line.

Syntax of Toolkit Client Resources

As mentioned above, X Toolkit applications (and Xt-based toolkit applications) are made up of predefined components called widgets. There can be widgets within widgets (e.g., a command button within a dialog box). The syntax of resource specifications for Toolkit clients parallels the levels of the widget hierarchy. Accordingly, you should think of a resource specification as having this format:

```
object.subobject[.subobject...].attribute: value
```

where:

object — is the client program or a specific instance of the program. (See "The –name Option" later in this chapter.)

subobjects — correspond to levels of the widget hierarchy (usually the major structures within an application, such as windows, menus, scrollbars, etc.).

*If a client was built with the X Toolkit, this should be noted on the reference page. In addition to certain application-specific resource variables, most clients that use the X Toolkit recognize a common set of resource variables, listed in Table 10-1.

In addition, X Toolkit clients recognize a set of Core resource variables, listed in Table G-1. However, though all Toolkit applications recognize these variables, not all applications make use of them. This fine distinction is addressed in Appendix G, *Widget Resources*, which gives a more technical discussion of how widgets use resources, and how applications use widgets. Appendix G also gives a detailed listing of the resources defined by each of the Athena widgets.

attribute is a feature of the last *subobject* (perhaps a command button), such as background color or a label that appears on it.

value is the actual setting of the resource *attribute*, i.e., the label text, color, or other feature.

The type of *value* to supply is often evident from the name of the resource or from the description of the resource variable on the reference page. Most of these values are similar to those used with the command-line options described in Chapter 10.

For example, various resources, such as `borderColor` or `background`, take color specifications; `geometry` takes a geometry string, `font` takes a font name, and so on. Logical values, such as the values taken by `scrollBar`, can generally be specified as: `on` or `off`; `yes` or `no`; or `True` or `False`.

Tight Bindings and Loose Bindings

Binding refers to the way in which components of a resource specification are linked together. Resource components can be linked in two ways:

- By a *tight* binding, represented by a dot (.).
- By a *loose* binding, represented by an asterisk (*).

A tight binding means that the components on either side of the dot must be next to one another in the widget hierarchy. A loose binding is signaled by an asterisk, a wildcard character which means there can be any number of levels in the hierarchy between the two surrounding components (including none).

If you want to specify tight bindings, you must be very familiar with the widget hierarchy: it's easy to use tight bindings incorrectly.

For example, this resource specification to request that *xterm* windows be created with a scrollbar doesn't work:

```
xterm.scrollBar:     True
```

The previous specification ignores the widget hierarchy of *xterm*, in which the VT102 window is considered to be one widget, the Tektronix window another, and the menus a third. This means that if you want to use tight bindings to request that *xterm* windows be created with a scrollbar, you should specify:

```
xterm.vt100.scrollBar:     True
```

Of course rather than decipher the widget hierarchy (which may even change with subsequent versions of an application), it is far simpler just to use the asterisk connector in the first place:

```
xterm*scrollBar:     True
```

Note that the asterisk is interpreted very differently in resource syntax than in the UNIX C shell. In the shell, the asterisk is a wildcard that can represent zero or more *characters*. In a resource file, the asterisk represents zero or more *complete components* in the resource name.

(Zero refers to the case in which the asterisk simply connects the previous and subsequent components.) Don't make the mistake of trying to use the asterisk to match *partial* component names. If you want to set the same specification for clients with similar names, you cannot use a common abbreviation. For example, if you would like *xcalc*, *xclock*, and *xclipboard* to display in reverse video, you can't write:

```
xc*reverseVideo:  True
```

In an application that supports multiple levels of widgets, you can mix asterisks and periods. In general, though, the developers of X recommend always using the asterisk rather than the dot as the connector even with simple applications, since this gives application developers the freedom to insert new levels in the hierarchy as they produce new releases of an application.

Instances and Classes

Each component of a resource specification has an associated *class*. Several different widgets, or widget attributes, may have the same class. For example, in the case of *xterm*, the color of text (`foreground`), the pointer color, and the text cursor color are all defined as *instances* of the class `Foreground`. This makes it possible to set the value of all three with a single resource specification. That is, if you wanted to make the text, the pointer, and the cursor dark blue, you could specify either:

```
xterm*foreground:      darkblue
xterm*cursorColor:     darkblue
xterm*pointerColor:    darkblue
```

or:

```
xterm*Foreground:      darkblue
```

Initial capitalization is used to distinguish class names from instance names. By convention, class names always begin with an uppercase letter, while instance names always begin with a lowercase letter. Note, however, that if an instance name is a compound word (such as `cursorColor`), the second word is usually capitalized.

The real power of class and instance naming is not apparent in applications such as *xterm* that have a simple widget hierarchy. In complex applications written with the X Toolkit or the Motif Toolkit, class and instance naming allows you to do such things as specify that all buttons in dialog box be blue but that one particular button be red. For example, in the hypothetical *xclient* application, you might have a resource file that reads:

```
xclient*buttonbox*Buttons*foreground:        blue
xclient*buttonbox*delete*foreground:         red
```

where `Buttons` is a class name and the `delete` button is an instance of the `Buttons` class. This type of specification works because an instance name always overrides the corresponding class name for that instance. Class names thus allow default values to be specified for all instances of a given type of object. Instance names can be used to specify exceptions to the rules outlined by the class names. Note that a class name can be used with a loose

binding to specify a resource for all clients. For example, this specification would say that the foreground colors for all clients should be blue:

```
*Foreground:        blue
```

The reference page for a given program should always give you both instance and class names for every resource variable you can set. You'll notice that in many cases the class name is identical to the instance name, with the exception of the initial capital letter. Often (but not always) this means that there is only one instance of that class. In other cases, the instance with the same name is simply the primary or most obvious instance of the class.

Wildcarding a Component Name with ?

As of Release 5, you can use a question mark (?) to represent *any single component* in a resource specification. (Naturally, you can't use a question mark as the final component, the resource variable itself.) The use of the question mark wildcard is a bit confusing and is best learned by example.

```
xclient.?.?.Background:    whitesmoke
```

The preceding line sets the background color for all widgets that are two subobjects below the application level. (You can also think of these subojects/widgets as the *grandchildren* of the top level window in the client's own window hierarchy. Typical "grandchildren" might be dialog boxes, menus, etc.)

Note also that the specification sets the background color for *only* those widgets. (The tight bindings ensure this.) A loose binding between the second question mark wildcard and the resource variable (`Background`) expands the coverage to include the second subobject level and also all further subobects:

```
xclient.?.?*Background:    whitesmoke
```

The use of the question mark requires a bit of finesse, but it simplifies specifications that have previously been very involved. Prior to Release 5, if you wanted to set the background color for grandchildren of the top level application window, you would have to provide a resource line for each one. Our first example line:

```
xclient.?.?.Background:    whitesmoke
```

does the same thing.

The use of the question mark and the asterisk wildcards together may confuse you, but there is an important distinction between them. The question mark always represents a single component. Thus, unless it is the first component, it must always be bracketed by connectors. (You'll generally use periods as the connectors between question mark wildcards to indicate a tight binding, but as we saw in the second example, sometimes an asterisk is the appropriate connector.) The most important thing to remember is that the presence of a question mark specifies that a component exists (though it does not specify the name of the component).

The asterisk specifies that zero or more (adjacent) components have been omitted from the resource. In effect, it is both a wildcard (that may represent adjacent components) and a connector.

The following section explains how the question mark wildcard is interpreted in determining the precedence of resource specifications. The section "Other Ways to Specify the Same Resource" (at the end of the *editres* tutorial) gives additional examples of how to use the question mark wildcard.

Precedence Rules for Resource Specification

Even within a single resource file, such as *.Xresources*, resource specifications often conflict. For instance, recall the example from the first page of the chapter involving the hypothetical *xclient* application:

```
xclient*Buttons.foreground:        blue
xclient*help.foreground:           red
```

The first resource specification makes the foreground color of all buttons (in the class `Buttons`) blue. The second resource specification overrides the first in one instance: it makes the foreground color of the `help` button (an instance of the class `Buttons`) red. In the event of conflicting specifications, there are a number of rules that the resource manager follows in deciding which resource specification should take effect.

We've already seen two of these rules, which are observable in the way the resource manager interprets definitions in a user-created resource file. (The first rule applies in the previous *xclient* example.)

- Instance names take precedence over class names.
- Tight bindings take precedence over loose bindings.

From just these two rules, we can deduce a general principle: the more specific a resource definition is, the more likely it is to be honored in the case of a conflict.

However, for cases in which you want to set things up very carefully, you should know a bit more about how programs interpret resource specifications.

For each resource, the program has both a complete, fully specified, tightly bound instance name and class name. In evaluating ambiguous specifications, the program compares the specification against both the full instance name and the full class name. If a component in the resource specification matches either name, it is accepted. If it matches more than one element in either name, it is evaluated according to these precedence rules:

1. The levels in the hierarchy specified by the user must match the program's expectations or the entry will be ignored. For example, if the program expects either:

   `xterm.vt100.scrollBar:` *`value`* *instance name*

   or:

   `XTerm.VT100.ScrollBar:` *`value`* *class name*

   the resource specification:

   `xterm.scrollBar:  True`

won't work, because the tight binding is incorrect. The objects `xterm` and `scrollBar` are not adjacent in the widget hierarchy: there is another widget, `vt100`, between them. The specification would work if you used a loose binding, however:

```
xterm*scrollBar:    True
```

(Note that the class name of *xterm* is `XTerm`, not `Xterm` as you might expect.) You might instead use the question mark wildcard to represent `vt100` in the widget hierarchy:

```
xterm.?.scrollBar:  True
```

This specification is perfectly valid. Note also that this line is more specific than (and thus takes precedence over) `xterm*scrollBar: True`.

2. Tight bindings take precedence over loose bindings. That is, entries with instance or class names prefixed by a dot are more specific than entries with names prefixed by an asterisk, and more specific entries take precedence. For example, the entry `xterm.vt100.geometry` will take precedence over the entry `xterm*geometry`.

3. Similarly, instances take precedence over classes. For example, the entry `*scrollBar` will take precedence over the entry `*Scrollbar`.

4. Left components carry more weight than right components. For example, the entry `xterm*background` will take precedence over `*background`.

5. An instance or class name that is explicitly stated takes precedence over one represented by the question mark wildcard, which in turn takes precedence over an omitted component. Thus, the specification `xterm*scrollbar` is more specific than `?*scrollBar`, which is still more specific than `*scrollBar`.

To illustrate these rules, let's consider the following resource specifications (shown in Example 11-1) for the hypothetical Toolkit application *xclient*.

Example 11-1. Sample resources

```
xclient.toc*Command.activeForeground:     black
*Command.Foreground:     green
```

Each of these lines specifies a foreground color. Both specifications are valid. Now, applying the rules of precedence, let's try to figure out what foreground color would be used for the *xclient* application's `include` command button. To determine how conflicting specifications are applied, the program tries to match these specifications against the complete tightly bound instance and class specifications. In this case, say the complete specifications are:

```
xclient.toc.messageFunctions.include.foreground          instance name
Xclient.Box.SubBox.Command.Foreground                    class name
```

Note that these specifications are the instance and class names for the same resource—which determines the foreground color of the `include` button. Each component of the instance name belongs to the class in the corresponding component of the class name. Thus, the instance `toc` occurs in the class `Box`, the `messageFunctions` instance name is from the class `SubBox`, etc. The `include` button is an instance of the `Command` class.

Both resource specification in Example 11-2 matches these instance and class names. However, with its tight bindings and instance names, `xclient.toc*Command.foreground` matches more explicitly (i.e., with higher precedence). The resource is set: the `foreground` color of the `include` button is `black`.

The specification `*Command.Foreground` also matches the instance and class names but is composed entirely of class names which are less specific; thus, it takes lower precedence than the first line in Example 11-2 (which sets the `include` button to `black`).

However, since the second line is also an acceptable specification, hypothetically it would set the foreground color of other objects in the `Command` class. Thus, if there were other *xclient* command buttons comparable to the `include` button in the hierarchy, this second line would set the foreground color of these buttons to `green`. Note that, since the line begins with the asterisk wildcard, the resource would be set for *xclient*, as well as any other application with a `Command` class.

Now let's consider some actual conflicting resource specifications and apply the rules of precedence. All three of the resources in Example 11-2 are valid specifications for the font of all instances of the class `Command` (without the jargon, the font for the labels on command buttons).

Example 11-2. What takes precedence?

```
*Command*Font:  -*-helvetica-bold-r-normal--*-120-*-*-*-*-iso8859-1
?*Command*Font:  7x14
XClipboard*Command*Font:  6x10
```

We've listed the resources in Example 11-2 in increasing order of specificity. Because of the initial loose binding, the first specification applies to any client with a command widget. The second specification also applies to any client with a command widget, but the introductory question mark (representing all clients) makes it more specific. Thus, the second line overrides the first and specifies that the font for command buttons for all applications is `7x14`.

The third line is even more specific because it begins with an actual class name (`XClipboard`) rather than the question mark wildcard. The third line specifies that the font for *xclipboard* command buttons is `6x10`. Note, however, that the second line still applies to all other clients—they will use the font `7x14` for command buttons. The third line simply introduces an exception.

If you want a more detailed description of how resource precedence works, see Chapter 9 of Volume Four, *X Toolkit Intrinsics Programming Manual*.

Some Common Resources

Each Toolkit command-line option (listed in Table 9-1) has a corresponding resource variable. Most X Toolkit (and Motif Toolkit) applications recognize some subset of these resources.

Table 11-1 lists the resource variables recognized by most Toolkit clients.

Table 11-1. Common Toolkit Resources

Instance Name	Class Name	Default	Description
`background`	`Background`	White	Background color
`foreground`	`Foreground`	Black	Foreground color
`borderColor`	`BorderColor`	Black	Border color
`borderWidth`	`BorderWidth`	1 pixel	Border width

Note that in a complex Toolkit application these values can occur at every level in a widget hierarchy. For example, our hypothetical *xclient* application might support these complete instance names:

```
xclient.background
xclient.buttonBox.background
xclient.buttonBox.commandButton.background
xclient.buttonBox.quit.background
```

These resources would specify the background color for the application window, the button-box area, any command buttons, and the quit command button, respectively.

Of course, the specification:

```
xclient*background
```

would match any and all of them.

Appendix G lists resources for each of the Athena widgets.

Event Translations

We've discussed the basics of resource naming syntax. From the sample resource settings, it appears that what many resource variables do is self-evident or nearly so. Among the less obvious resource variables, there is one type of specification, an event translation, that can be used with many clients and warrants somewhat closer examination.

User input and several other types of information pass from the server to a client in the form of *events*. An event is a packet of information that tells the client something it needs to act on, such as keyboard input. As mentioned in Chapter 1, *An Introduction to the X Window System*, moving the pointer or pressing a key, etc., causes *input* events to occur. When a program receives a meaningful event, it responds with some sort of action.

For many clients, the resource manager recognizes mappings between certain input events (such as a pointer button click) and some sort of action by the client program (such as selecting text). A mapping between one or more events and an action is called a *translation*. A resource containing a list of translations is called a *translation table*.

Many event translations are programmed into an application and are invisible to the user.* For our purposes we are only concerned with very visible translations of certain input events, primarily the translation of keystrokes and pointer button clicks to particular actions by a client program.

The Syntax of Event Translations

The operation of many clients, notably *xterm*, is partly determined by default input event translations. For example, as explained in Chapter 5, *The xterm Terminal Emulator*, selecting text with the first pointer button (an event) saves that text into memory (an action).

In this case, the input "event" is actually three separate X events:

1. Pressing the first pointer button.
2. Moving the pointer while holding down the first button.
3. Releasing the button.

Each of these input events performs a part of the action of selecting text:

1. Unselects any previously selected text and begins selecting new text.
2. Extends the selection.
3. Ends the selection, saving the text into memory (both as the PRIMARY selection and CUT_BUFFER0).

The event and action mappings would be expressed in a translation table as:

```
<Btn1Down>: select-start()\n\
<Btn1Motion>: select-extend()\n\
<Btn1Up>: select-end(PRIMARY,CUT_BUFFER0)
```

where each event is enclosed in angle brackets (<>) and produces the action that follows the colon (:). A space or tab generally precedes the action, though this is not mandatory:

```
<event>: action
```

A translation table must be a continuous string. In order to link multiple mappings as a continuous string, each event-action line should be terminated by a newline character (\n), which is in turn followed by a backslash (\) to escape the actual newline.

These are default translations for *xterm*.† All of the events are simple, comprised of a single button motion. As we'll see, events can also have modifiers: i.e., additional button motions or keystrokes (often Control or Meta) that must be performed with the primary event to pro-

*For more information on events and translations, see Volume Four, *X Toolkit Intrinsics Programming Manual.*
†They are actually slightly simplified versions of default translations. Before you can understand the actual translations listed on the *xterm* reference page in Part Three of this guide, you must learn more about the syntax of translations. In addition to the current chapter, read Appendix F, *Translation Table Syntax.*

duce the action. (Events can also have modifiers that *must not* accompany the primary event if the action is to take place.)

As you can see, the default actions listed in the table are hardly intuitive. The event-action mappings that can be modified using translation resources are usually described on the reference page for the particular client.

You can specify non-default translations using a translation table (a resource containing a list of translations). Since actions are part of the client application and cannot be modified, what you are actually doing is specifying alternative events to perform an action.* Keep in mind that only applications written with the X Toolkit (or an Xt-based toolkit such as the Motif Toolkit) recognize translation table syntax.

The basic syntax for specifying a translation table as a resource is:

```
[object*[subobject...]]*translations:   #override\
     [modifier]<event>:   action
```

The first line is basically like any other resource specification with a few exceptions. First, the final *argument* is always `translations`, indicating that one (or more) of the event-action bindings associated with the [*object*`*`[*subobject* . . .]] are being modified.

Second, note that `#override` is not the *value* of the resource; it is literal and indicates that what follows should override any default translations. In effect, `#override` is no more than a pointer to the true *value* of the resource: a new event-action mapping (on the following line), where the event may take a modifier.†

A not-so-obvious principle behind overriding translations is that you only literally "override" a default translation when the event(s) of the new translation match the event(s) of a default translation *exactly*. If the new translation does not conflict with any existing translation, it is merely appended to the defaults.

In order to be specified as a resource, a translation table must be a single string. The `#override` is followed by a backslash (\) to indicate that the subsequent line should be a continuation of the first.

In the previous basic syntax example, the *value* is a single event-action mapping. The *value* could also be a list of several mappings, linked by the characters "\n\" to make the resource a continuous string.

The following *xterm* translation table shows multiple event-action mappings linked in this manner:

```
*VT100.Translations:   #override\
     <Btn1Down>:     select-start()\n\
     <Btn1Motion>:   select-extend()\n\
     <Btn1Up>:       select-end(PRIMARY,CUT_BUFFER0)
```

*As we'll see, in certain cases you may be able to supply an alternative *argument* (such as a selection name) to an action. These changes *are* interpreted by the resource manager.

†The use of modifiers can actually become quite complicated, sometimes involving multiple modifiers. For our purposes, we'll deal only with simple modifiers. For more information on modifiers, see Appendix F in this guide and Volume Four, *X Toolkit Intrinsics Programming Manual.*

xterm Translations to Use xclipboard

As explained in Chapter 5, the *xclipboard* client provides a window in which you can store text selected from other windows. You can also paste text from the *xclipboard* window into other windows. See the discussion of *xclipboard* in Chapter 5 before proceeding.

You can specify translations for *xterm* so that text you copy with the pointer is made the CLIPBOARD selection. The CLIPBOARD selection is the property of the *xclipboard* client. If you are running *xclipboard* and you copy text to be made the CLIPBOARD selection, this text automatically appears in the *xclipboard* window.

Some sample translations that would allow you to use the *xclipboard* in this way are:

```
*VT100.Translations: #override\
     Button1 <Btn3Down>: select-end(PRIMARY,CUT_BUFFER0,CLIPBOARD)\n\
     !Shift <Btn2Up>:        insert-selection(CLIPBOARD)\n\
     ~Shift ~Ctrl ~Meta <Btn2Up>:   insert-selection(PRIMARY,CUT_BUFFER0)
```


According to this translation table, while selecting text with `Button1` (the modifier), the event of pressing the third pointer button (`Btn3Down`), while continuing to hold down the first button, produces the action of making the text the CLIPBOARD selection (as well as making it the PRIMARY selection and saving it to CUT_BUFFER0). Basically, we've taken the default `select-end` translation—which uses the first pointer button and the arguments `PRIMARY`, `CUT_BUFFER0`—and added the `Btn3Down` action and the `CLIPBOARD` argument.

The second line specifies a way to paste the CLIPBOARD selection: by holding the Shift key and clicking the second pointer button.

The third line modifies the way the contents of the PRIMARY selection or CUT_BUFFER0 are pasted into a window. As described in Chapter 5, by default pressing the second pointer button pastes the contents of the PRIMARY selection. If there is no PRIMARY selection, the contents of the cut buffer are pasted. The default translation that sets this behavior is the following:

```
~Ctrl ~Meta <Btn2Up>:     insert-selection(PRIMARY,CUT_BUFFER0)
```

This translation specifies that clicking (actually releasing) pointer button 2 (while pressing any modifier button or key *other than Control or Meta*) performs the `insert-selection` action. The arguments to `insert-selection` indicate that this action inserts text from the PRIMARY selection or, if the selection is empty, from cut buffer 0. Excluding the Control and Meta keys is intended to prevent conflict with other action mappings.* We've added `~Shift` to prevent a conflict with the action that pastes the CLIPBOARD selection.

Thus, according to the translations in the example, if you select text as usual with the first pointer button, and then additionally press the third button (while continuing to hold down the first button), the text becomes the CLIPBOARD selection and appears automatically in the *xclipboard* window, as shown in Figure 11-1.

*`~Ctrl` is specified to keep this translation from conflicting with the translations that invoke the *xterm* menus; `~Meta` prevents a conflict with *twm* functions. (As you may recall, *twm* is the window manager MIT ships with the standard version of X.)

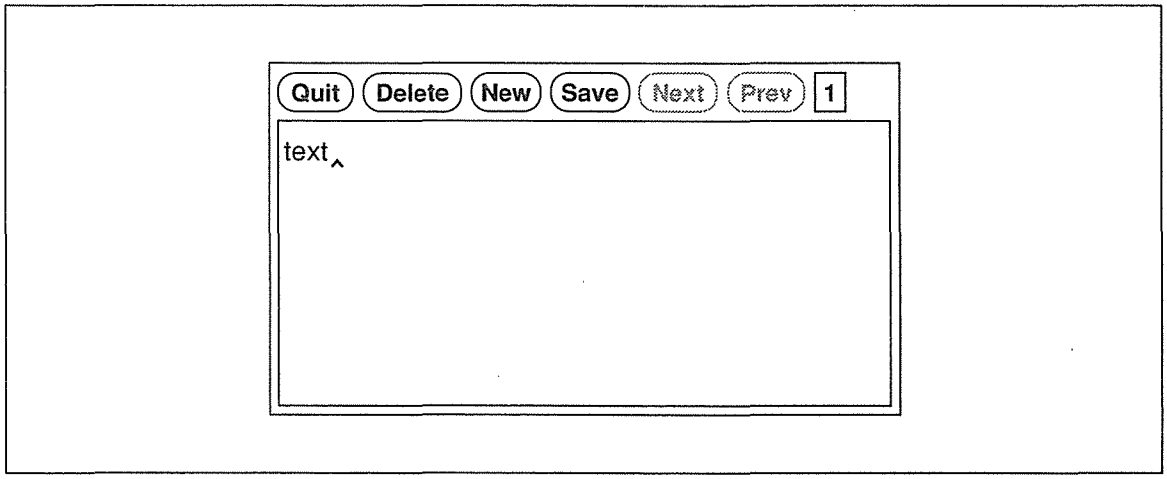

Figure 11-1. Selected text appears automatically in the xclipboard window

Since our first translation specifies a different event/action mapping than the default translation for selecting text (discussed in the previous section), the default translation still applies. If you select text with the first pointer button alone, that text is still made the PRIMARY selection and fills CUT_BUFFER0. To send text to the *xclipboard*, you would need to press the third pointer button as well; thus, not all selected text needs to be made the CLIPBOARD selection (and sent automatically to the *xclipboard*).

There are advantages to making only certain selections CLIPBOARD selections. You can keep *xclipboard* running and make many text selections by the default method (first pointer button), without filling up the *xclipboard* window. And chances are you don't want to save every piece of text you copy for an extended period of time, anyway.

The CLIPBOARD selection and the *xclipboard* client also get around the potential problems of selection ownership discussed in Chapter 5. Once text becomes the CLIPBOARD selection, it is owned by the *xclipboard* client. Thus, if the client from which text was copied (the original owner) goes away, the selection is still available, owned by the *xclipboard*, and can be transferred to another window (and translated to another format if necessary).

Entering Frequently Used Commands with Function Keys

The sample *xterm* translations to use the *xclipboard* client involve just a few of the actions *xterm* recognizes. Among the more useful translations you can specify for *xterm* are function key mappings that allow you to enter frequently used commands with a single keystroke. This sort of mapping involves an action called `string`, which passes a text string to the shell running in the *xterm* window.

The translation table syntax for such a function key mapping is fairly simple. The following line maps the text string "lpq –Pprinter1" (the BSD 4.3 command to check the queue for the printer named printer1) to the F1 function key:

```
<Key>F1:        string("lpq -Pprinter1")
```

Notice the quotes surrounding the text string. If the argument to `string` includes spaces or non-alphanumeric characters, the whole argument must be enclosed in one pair of double quotes. (Don't make the mistake of quoting individual words.)

The translation table would be:

```
*VT100.Translations: #override\
        <Key>F1:        string("lpq -Pprinter1")
```

This sample translation causes `lpq -Pprinter1` to be passed to the command line in the active *xterm* window when you press the F1 function key, as in Figure 11-2.

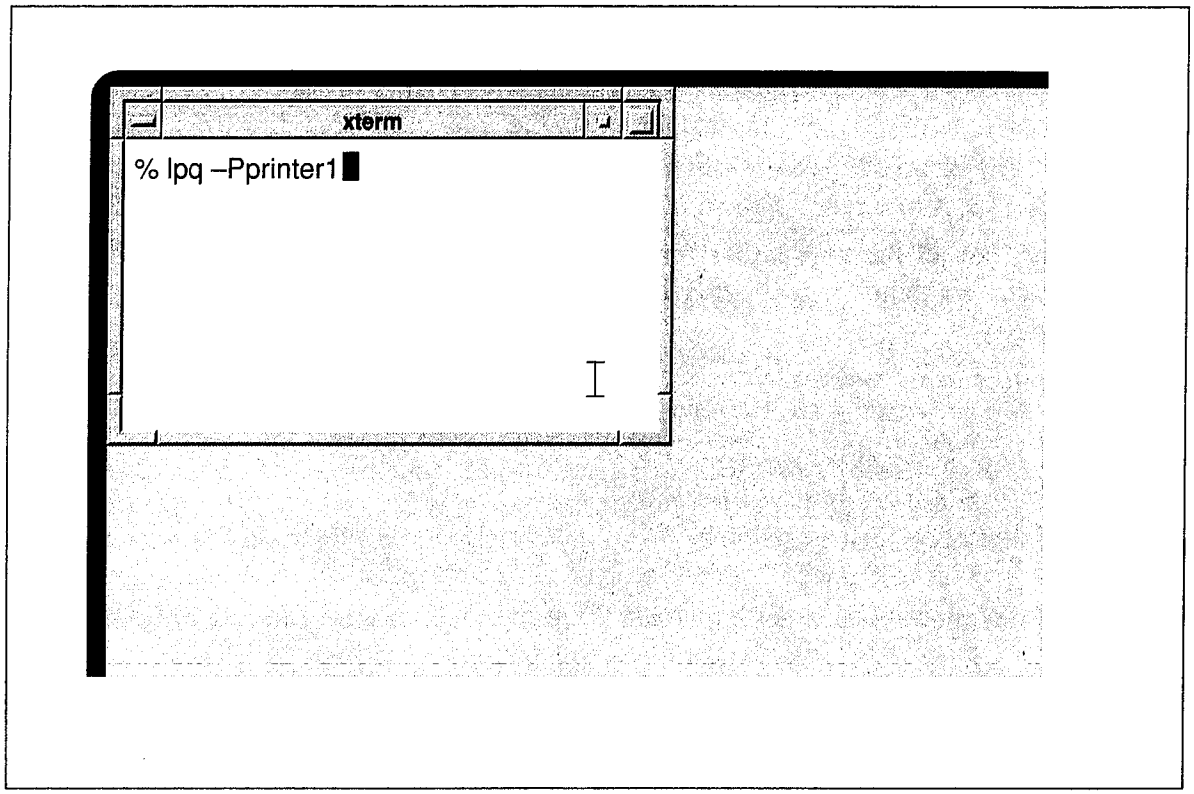

Figure 11-2. Pushing F1 passes command text to xterm shell

Notice, however, that the command is not invoked because there has been no carriage return. The sample translation does not specify a return. You can add a return as the argument to another `string` action within the same translation.

To specify the Return (or any) key, use the hexadecimal code for that key as the argument to `string`. Keycodes and the procedure for determining them are explained in Chapter 14, *Setup Clients*. The letters "0x" signal a hexadecimal key code. If you want to enter a

key as an argument to `string`, use "0x" followed by the specific code. The code for the Return key is "d" or "0d."* The following translation table specifies that pressing F1 passes the line `lpq -Pprinter1` followed by a carriage return to an *xterm* window:

```
*VT100.Translations: #override\
        <Key>F1:        string("lpq -Pprinter1") string(0x0d)
```

Remember, you can list several translations in a single table. The following table maps function keys F1 through F3:

```
*VT100.Translations: #override\
        <Key>F1:        string("lpq -Pprinter1") string(0x0d)\n\
        <Key>F2:        string("cd ~/bitmap;ls") string(0x0d)\n\
        <Key>F3:        string("cd /usr/lib/X11") string(0x0d)
```

According to these translations, pressing F2 inserts the command string `cd ~/bitmap;ls`, which changes directory to *~/bitmap* and then lists the contents of that directory. Notice that you can issue multiple commands (*cd*, *ls*) with a single key. Pressing F3 changes directory to */usr/lib/X11*.

Keep in mind that all the translations for an application can appear in the same table. For example, we can combine the *xterm* translations to use the *xclipboard* with the translations to map function keys.

```
*VT100.Translations: #override\
     Button1 <Btn3Down>: select-end(PRIMARY,CUT_BUFFER0,CLIPBOARD)\n\
     !Shift <Btn2Up>:        insert-selection(CLIPBOARD)\n\
     ~Shift ~Ctrl ~Meta <Btn2Up>:    insert-selection(PRIMARY,CUT_BUFFER0)
     <Key>F1:                string("lpq -Pprinter1") string(0x0d)\n\
     <Key>F2:                string("cd ~/bitmap;ls") string(0x0d)\n\
     <Key>F3:                string("cd /usr/lib/X11") string(0x0d)
```

The order of the translations is not important. However, it is necessary to end all but the final line with the sequence "\n\" to make the resource a continuous string.

Other Clients that Recognize Translations

xterm is not the only client whose operation can be modified by specifying event translations as resources (though it is probably the client you'll be most interested in modifying). Among the standard clients, *xbiff*, *xcalc*, *xdm*, *xman*, and *xmh* all recognize certain actions that can be mapped to particular keys or key combinations using the translation mechanism. See the relevant client reference pages in Part Three of this guide for complete lists of actions.

You can also modify the operation of the Text widget used by *xedit*, *xmh*, and other X Toolkit applications. See the *xedit* reference page in Part Three for a list of actions recognized by the Text widget. Keep in mind, however, that the default Text widget recognizes dozens of com-

*As explained in Chapter 14, *Setup Clients*, the command `xmodmap -pk` returns a long listing of all keycodes. The codes have the either of the following forms:

0xff*ab*
0x00*ab*

where *ab* represents two alphanumeric characters. To specify a key as an argument to `string`, you can omit the "ff" or "00" in the *xmodmap* listing.

mands, which are summarized in the discussion of *xedit* in Chapter 8, *Other Clients*. It may not be practical or desirable to modify them all.

If you choose to modify the Text widget, you can do so for all relevant clients by introducing the translations with the line:

```
*Text*Translations: #override\
```

You can also specify different translations for different clients that use the widget by prepending the client's name. To affect the operation of the Text widget only under *xedit*, introduce the translation table with the line:

```
Xedit*Text*Translations: #override\
```

In modifying the operation of the Text widget, keep in mind that insert mode is the default. In other words, like *emacs*, most of the individual keystrokes you type are added to the text file; an exception is Backspace, which predictably deletes the preceding character. The commands to move around in a file, copy and delete text, etc., involve a combination of keys, one of which is generally a modifier key. If you want to modify a command, you should use an alternative key combination, rather than a single key.

For example, the following table offers two suitable translations:

```
*Text*Translations: #override\
      Meta<Key>f:     next-page()\n\
      Meta<Key>b:     previous-page()
```

The first translation specifies that pressing the key combination Meta-f moves the cursor ahead one page in the file (scrolls the file forward one window); the second translation specifies that Meta-b moves the cursor back one page. The actions performed are fairly obvious from their names. For a complete list of actions recognized by the Text widget, see the *xedit* reference page.

For more information about events, actions, and translation table syntax, see Appendix F, *Translation Table Syntax*, and Volume Four, *X Toolkit Programming Manual*.

Though *mwm* does not provide actions that can be modified using a translation table, you can change the key and pointer button events used to invoke window manager functions by editing a special file called *.mwmrc* in your home directory. See Chapter 13, *Customizing mwm*, for details.

How to Set Resources

Learning to write resource specifications is a fairly manageable task, once you understand the basic rules of syntax and precedence. In contrast, the multiple ways you can set resources—for a single system, for multiple systems, for a single user, for all users—can be confusing. For our purposes, we are primarily concerned with specifying resources for a single user running applications both on the local system and on remote systems in a network.

As we've said, resources are generally specified in files. A resource file can have any name you like. Resources are generally "loaded" into the X server by the *xrdb* client, which is normally run from your startup file or run automatically by *xdm* when you log in. (See Appendix A, *Managing Your Environment*, for information about startup files and *xdm*.) Prior to Release 2 of X, there was only one resource file called *.Xdefaults*, placed in the user's home directory. If no resource file is loaded into the server by *xrdb*, the *.Xdefaults* file will still be read.

Remember that X allows clients to run on different machines across a network, not just on the machine that supports the X server. The problem with the older *.Xdefaults* mechanism was that users who were running clients on multiple machines had to maintain multiple *.Xdefaults* files, one on each machine. By contrast, *xrdb* stores the application resources directly in the server, thus making them available to all clients, regardless of the machine on which the clients are running. As we'll see, *xrdb* also allows you to change resources without editing files.

Of course, you may want certain resources to be set on all machines and others to be set only on particular machines. See the section "Other Sources of Resource Definition" later in this chapter for information on setting machine-specific resources. This section gives an overview of additional ways to specify resources using a variety of system files.

In addition to loading resource files, you can specify defaults for a particular instance of an application from the command line using two options: `-xrm` and `-name`.

First we'll consider a sample resources file. Then we'll take a look at the use of the `-xrm` and `-name` command-line options. Finally, we'll discuss various ways you can load resources using the *xrdb* program and consider other sources of resource definition, later in this chapter.

A Sample Resources File

Figure 11-3 shows a sample resources file. This file sets the border width for all clients to a default value of two pixels, and sets other specific variables for *xclock* and *xterm*. The meaning of each variable is obvious from its name (for example, `xterm*scrollBar: True` means that *xterm* windows should be created with a scrollbar.

Note that comments are preceded by an exclamation point (!).

For a detailed description of each possible variable, see the appropriate client reference pages in Part Three of this guide.

```
*borderWidth:          2
!
! xclock resources
!
xclock*borderWidth:    5
xclock*geometry:       64x64
!
! xterm resources
!
xterm*curses:          on
xterm*cursorColor:     skyblue
xterm*pointerShape:    pirate
xterm*jumpScroll:      on
xterm*saveLines:       300
xterm*scrollBar:       True
xterm*scrollKey:       on
xterm*background:      black
xterm*borderColor:     blue
xterm*borderWidth:     3
xterm*foreground:      white
xterm*font:            8x13
```

Figure 11-3. A sample resources file

Specifying Resources from the Command Line

Two command-line options supported by all clients written with the X Toolkit can be useful in specifying resources.

The –xrm Option

The `-xrm` option allows you to set on the command line any specification that you would otherwise put into a resources file. For example:

```
% xterm -xrm 'xterm*Foreground: blue' &
```

Note that a resource specification on the command line must be quoted using the single quotes in the line above.

The `-xrm` option only specifies the resource(s) for the current instance of the application. Resources specified in this way do not become part of the resource database.

The `-xrm` option is most useful for setting classes, since most clients have command-line options that correspond to instance variable names. For example, the `-fg` command-line option sets the `foreground` attribute of a window, but `-xrm` must be used to set `Foreground`.

Note also that a resource specified with the `-xrm` option will not take effect if a resource that takes precedence has already been loaded with *xrdb*. For example, say you've loaded a resource file that includes the specification:

```
xterm*pointerShape:  pirate
```

The command-line specification of another cursor will fail:

```
% xterm -xrm '*pointerShape:  gumby' &
```

because the resource `xterm*pointerShape` is more specific than the resource `*pointerShape`. Instead, you'll get an *xterm* with the previously specified pirate cursor.

To override the resource database (and get the Gumby cursor), you'd need to use a resource equally (or more) specific, such as the following:

```
% xterm -xrm 'xterm*pointerShape:  gumby' &
```

How –name Affects Resources

The `-name` option lets you name one instance of an application; the server identifies the single instance of the application by this name. The name of an application affects how resources are interpreted.

For example, the following command sets the *xterm* instance name to `bigxterm`:

```
% xterm -name bigxterm &
```

When this command is run, the client uses any resources specified for `bigxterm` rather than for `xterm`.

The `-name` option allows you to create different instances of the same application, each using different resources. For example, you could put the following entries into a resource file such as *.Xresources*:

```
XTerm*Font:           8x13
smallxterm*Font:      6x10
smallxterm*Geometry: 80x10
bigxterm*Font:        9x15
bigxterm*Geometry:    80x55
```

You could then use these commands to create *xterm*s of different specifications:

```
% xterm &
```

would create an *xterm* with the default specifications, while:

```
% xterm -name bigxterm &
```

would create a big *xterm*, 80 characters across by 55 lines down, displaying in the font `9x15`. The command:

```
% xterm -name smallxterm &
```

would create a small *xterm*, 80 characters across by 10 lines down, displaying in the font `6x10`.

Setting Resources with xrdb

The *xrdb* program saves you from the difficulty of maintaining multiple resource files if you run clients on multiple machines. It stores resources in the X server, where they are accessible to all clients using that server. (Technically speaking, the values of variables are stored in a data structure referred to as the RESOURCE_MANAGER property of the root window of screen 0 for that server. From time to time, we may refer to this property simply as the resource database.)

The appropriate *xrdb* command line should normally be placed in your *.xinitrc* file or *.xsession* file to initialize resources at login, although it can also be invoked interactively. It has the following syntax:

xrdb [*options*] [*filename*]

The *xrdb* client takes several options, all of which are documented on the reference page in Part Three of this guide. Several of the most useful options are discussed in subsequent sections. (Those that are not discussed here have to do with *xrdb*'s ability to interpret C preprocessor-style defined symbols; this is an advanced topic. For more information, see the *xrdb* reference page in Part Three of this guide, and the *cpp*(1) reference page in your *UNIX Reference Manual.*)

The optional *filename* argument specifies the name of a file from which the values of client variables (resources) will be read. If no filename is specified, *xrdb* will expect to read its data from standard input. That is, the program will appear to hang, until you type some data, followed by an end-of-file (Control-D on many UNIX systems). Note that whatever you type will override the previous contents of the RESOURCE_MANAGER property, so if you inadvertently type *xrdb* without a filename argument, and then quit with Control-D, you will delete any previous values. (You can append new settings to current ones using the `-merge` option discussed later in this chapter.)

The resource *filename* can be anything you want. Two commonly used names are *.Xresources* and *.Xdefaults*.

You should load a resource file with the *xrdb* `-load` option. For example, to load the contents of your *.Xresources* file into the RESOURCE_MANAGER, you would type:

```
% xrdb -load .Xresources
```

Querying the Resource Database

You can find out what options are currently set by using the `-query` option. For example:

```
% xrdb -query
XTerm*ScrollBar:      True
bigxterm*font:        9x15
bigxterm*Geometry:    80x55
smallxterm*Font:      6x10
smallxterm*Geometry:  80x10
xterm*borderWidth:    3
```

If *xrdb* has not been run, this command will produce no output.

Loading New Values into the Resource Database

By default, *xrdb* reads its input (either a file or standard input) and stores the results into the resource database, replacing the previous values. If you simply want to merge new values with the currently active ones (perhaps by specifying a single value from standard input), you can use the `-merge` option. Only the new values will be changed; variables that were already set will be preserved rather than overwritten with empty values.

For example, let's say you wanted to add new resources listed in the file *new.values*. You could say:

```
% xrdb -merge new.values
```

As another example, if you wanted all subsequently run *xterm* windows to have scrollbars, you could use standard input, and enter:

```
% xrdb -merge
xterm*scrollBar:    True
```

and then press Control-D to end the standard input. Note that because of precedence rules for resource naming, you may not automatically get what you want. For example, if you specify:

```
xterm*scrollBar:      True
```

and the more specific value:

```
xterm*vt100.scrollBar:      False
```

has already been set, your new, less specific setting will be ignored. The problem isn't that you used the `-merge` option incorrectly—you just got caught by the rules of precedence.

If your specifications don't seem to work, use the `-query` option to list the values in the RESOURCE_MANAGER property and look for conflicting specifications.

Note also that when you add new specifications, they won't affect any programs already running, but only programs started after the new resource specifications are in effect. (This is also true even if you overwrite the existing specifications by loading a new resource file. Only programs run after this point will reflect the new specifications.)

Saving Active Resource Definitions in a File

Assume that you've loaded the RESOURCE_MANAGER property from an *.Xresources* or other file. However, you've dynamically loaded a different value using the `-merge` option and you'd like to make the new value your default.

You don't need to edit the file manually (although you certainly could.) The `-edit` option allows you to write the current value of the RESOURCE_MANAGER property to a file. If the file already exists, it is overwritten with the new values. However, *xrdb* is smart enough to

preserve any comments and preprocessor declarations in the file being overwritten, replacing only the resource definitions.

For example:

```
% xrdb -edit ~/.Xresources
```

will save the current contents of the RESOURCE_MANAGER property in the file *.Xresources* in your home directory.

If you want to save a backup copy of an existing file, use the `-backup` option:

```
% xrdb -edit .mydefaults -backup old
```

The string following the `-backup` option is used as an extension to be appended to the old filename. In the prior example, the previous copy of *.mydefaults* would be saved as *.mydefaults.old*.

Removing Resource Definitions

You can delete the definition of the RESOURCE_MANAGER property from the server by calling *xrdb* with the `-remove` option.

There is no way to delete a single resource definition other than to read the current *xrdb* values into a file. For example:

```
% xrdb -query > filename
```

Use an editor to edit the file, deleting the resource definitions you no longer want and save the file:

```
% vi filename
```

Then read the edited values back into the RESOURCE_MANAGER with *xrdb*:

```
% xrdb -load filename
```

Listing the Current Resources for a Client: appres

The *appres* (*app*lication *res*ource) program lists the resources that currently might apply to a client. These resources may be derived from several sources, including the user's *.Xresources* file and a system-wide application defaults file. The directory */usr/lib/X11/app-defaults* contains application default files for several clients. The function of these files is discussed in the next section. For now, be aware that all of the resources contained in these files begin with the class name of the application.

Also be aware that *appres* has one serious limitation: it cannot distinguish between valid and invalid resource specifications. It lists all resources that might apply to a client, whether or not the resources are correctly specified.

appres lists the resources that apply to a client having the *class_name* and/or *instance_name* you specify. Typically, you would use *appres* before running a client program to find out what resources the client program will access.

For example, say you want to run *xterm* but you can't remember the latest resources you've specified for it, whether you've loaded them, or perhaps what some of the application defaults are, etc. You can use the *appres* client to check the current *xterm* resources. If you specify only a class name, as in this command line:

```
% appres XTerm
```

appres lists the resources that any *xterm* would load. In the case of *xterm*, this is an extensive list, encompassing all of the system-wide application defaults as well as any other defaults you have specified in a resource file.

You can additionally specify an instance name to list the resources applying to a particular instance of the client, as in:

```
% appres XTerm bigxterm
```

If you omit the class name, *xappres* assumes the class `-NoSuchClass-`, which has no defaults, and returns only the resources that would be loaded by the particular instance of the client.

Note that the instance can simply be the client name, for example, `xterm`. In that case none of the system-wide application defaults would be listed, since all begin with the class name `XTerm`. For example, the command:

```
% appres xterm
```

might return resources settings similar to these:

```
xterm.vt100.scrollBar:  True
xterm*PhonyResource:    youbet
xterm*pointerShape:     gumby
xterm*iconGeometry:     +50+50
*VT100.Translations:    #override\
    Button1 <Btn3Down>:   select-end(CLIPBOARD)\n\
    ~Ctrl ~Meta <Btn2Up>: insert-selection(PRIMARY,CLIPBOARD)
```

Most of these resources set obvious features of *xterm*. The translation table sets up *xterm* to use the *xclipboard*. Notice also that *appres* has returned an invalid resource called `Phony-Resource` that we created for demonstration purposes. You can't rely on *appres* to tell you what resources a client will actually load because the *appres* program cannot distinguish a valid resource specification from an invalid one. Still, it can be fairly useful to jog your memory as to the defaults you've specified in your *.Xresources* file, as well as the system-wide application defaults.

Other Sources of Resource Definition

If *xrdb* has not been run, the RESOURCE_MANAGER property will not be set. Instead, the resource manager looks for a file called *.Xdefaults* in the user's home directory. As we discussed earlier, resources found in this way are only available to clients running on the local machine.

Whether or not resources have been loaded with *xrdb*, when a client is run these sources of resource definition are consulted in this order:

1. The client's application defaults file(s) (if any), which usually reside in the directory */usr/lib/X11/app-defaults*, will be loaded into the resource manager. (Note that the path can be reset with the XFILESEARCHPATH environment variable.) Application-specific resource files generally have the name *Class*, where *Class* is the class name of the client program.

   Any other application-specific resource files: a resource file named by the variable XUSERFILESEARCHPATH; or if this variable is not set, a file in the directory named by the environment variable XAPPLRESDIR.

2. Resources loaded into the RESOURCE_MANAGER property of the root window with *xrdb*; these resources are accessible regardless of the machine on which the client is running.

   If no resources are loaded in this way, the resource manager looks for an *.Xdefaults* file in the user's home directory; these resources are only available on the local machine.

3. Screen-specific resources loaded into the SCREEN_RESOURCES property of the root window with *xrdb*. The resource manager will sort and place the resources in RESOURCE_MANAGER (where they will apply to all screens) or in SCREEN_RESOURCES (where they will apply to the appropriate screen).

4. Next, the contents of any file specified by the shell environment variable XENVIRONMENT will be loaded.

   If this variable is not defined, the resource manager looks for a file named *.Xdefaults-**hostname*** in the user's home directory, where ***hostname*** is the name of the host where the client is running.

   These methods are used to set user- and machine-specific resources.

5. Any values specified on the command line with the `-xrm` option will be loaded for that instance of the program.

The resource specifications from these various sources will be loaded and merged according to the precedence rules described earlier in the section "Precedence Rules for Resource Specification."

The client will then merge these various defaults specified by the user with its own internal defaults, if any.

Finally, if the user has specified any options on the command line (other than with the `-xrm` option), these values will override those specified by resource defaults, regardless of their source.

Setting Resources for Color Versus Monochrome Screens

Chances are you would specify different resources for a client running on a color screen than for a client running on a monochrome screen. Prior to Release 5, the resource manager could not apply different sets of color resources for different screen types. Release 5 offers two innovations to help users specify resources for both color and monochrome displays:

1. A resource variable called `customization` (class `Customization`), which can be used to invoke a special set of application defaults for color or monochrome screens.
2. Screen-specific resource databases. The *xrdb* program has been updated to sort resources according to screen specifics, making certain resources available on a per screen basis and other available globally.

The first innovation relies on the existence of customized application defaults files. A few of the standard Release 5 clients come with additional defaults files suitable for color screens; the system administrator would have to create them for other clients.

You can apply these customized application defaults files—or specify your own special defaults—on a per screen basis (number 2 above) using a particular syntax in your *.Xresources* file.

The following two sections explain how to access the available customized app-defaults files and how to set your own resource file so that custom defaults (the system's or your own) are applied in the appropriate circumstances.

Loading Custom Application Defaults Files

As introduced in Chapter 3, *Working in the X Environment*, several clients have so-called *application defaults files* that provide resource definitions for certain client features. The defaults used on a particular system combine with other factors (program internals, user-supplied resources, and command-line options, etc.) to determine how the client looks and behaves.

Application defaults files are generally kept in the directory */usr/lib/X11/app-defaults*. (This path can be reset using the XFILESEARCHPATH environment variable, but setting up and maintaining the app-defaults directory is usually the system administrator's responsibility.) App-defaults files are often named ***Class*** to match the class name of the client, but there are exceptions.

In order to deal with the limitations of having a single app-defaults file when clients might be displayed on either a color or monochrome screen, X developers have added the `customization` resource in R5. You can use this resource to specify an alternative application defaults file. Commonly this alternative file will be a list of color resources. The naming convention for a color app-defaults file is the name of the standard app-defaults file (generally the class name) followed by a hyphen and the word "color" (i.e., ***Class-color***). A few standard R5 clients come with both a standard app-defaults file and also an alternative color file:

- bitmap
- editres
- xcalc
- xlogo

The *oclock* client comes with only one app-defaults file—called *Clock-color*. Note that this is an exception to the naming convention. *Clock* is the class name of the widget around which the *oclock* client is built.

With the exception of *oclock*, all of the app-defaults filenames follow the conventions outlined previously. Thus, the standard app-defaults file for *bitmap* is called *Bitmap* and the color file is called *Bitmap-color*. *xcalc* is shipped with two files called *XCalc* and *XCalc-color*, etc.

You invoke the color defaults for a single instance of a client program by using the `-customization` resource with the `-xrm` option and supplying the resource value `-color`, as in the following example:

```
bitmap -xrm "*customization:  -color" &
```

This command creates a *bitmap* window that uses the color defaults in *Bitmap-color*. This file provides the most vivid defaults of all the files currently included in the standard distribution.

The "color" defaults for *xcalc* are particularly uninteresting—all black, white and gray! Run the following command if you want to see for yourself.

```
xcalc -xrm "*customization:  -color" &
```

Depending on the particular defaults, if you run a "colorized" version of a client on a monochrome (or grayscale) screen, you may not be able to see all of the window's features. *xcalc* seems to be an exception. Even with color defaults, it should work fine on most screens.

When a client has both a standard and a custom defaults file, they usually do not have any conflicting specifications. In many cases, the custom file will begin with a line that invokes the standard file, which contains the more "global" defaults that can be applied regardless of the screen. The following line appears as the first specification in *Bitmap-color*, invoking the standard app-defaults:

```
#include "Bitmap"
```

It is possible for the system administrator to create files with defaults intended for monochrome screens. These files should generally be called ***Class-mono***. You can then supply

-mono as the value to the customization resource:

```
bitmap -xrm "*customization:  -mono" &
```

A monochrome defaults file might be simply a symbolic link to the regular app-defaults file or it might provide other specifications. Note, however, that Release 5 of X provides no such "–mono" files.

Thus far, we've shown you how to access the customized resource files on your system from the command line. The next section describes how to make certain resources apply depending on the kind of screen you're using.

Setting Screen-specific Resources

If you would like certain color (or mono) defaults to be used whenever you're working on a color (or mono) screen, you can edit your *.Xresources* file so this happens. These screen-specific resources may be accessed from custom defaults files or may be settings you write yourself.

The following lines specify that *bitmap* should be run using the color app-defaults file *when the screen is color*. (The line `#ifdef COLOR` establishes this condition; `#endif` marks the end of the conditional.)

```
#ifdef COLOR
bitmap*customization:  -color
#endif
```

Remember that the file *Bitmap-color* begins by "including" the standard defaults file (*Bitmap*). When you use *xrdb* to load a resource file that conditionally calls *Bitmap-color*, the resource manager sorts those resources that rely on a color monitor from those that can be applied regardless of the monitor (i.e., globally). When you subsequently run the client, you get the defaults appropriate for the monitor you are using!

If you want all existing "–color" app-defaults files to be used, omit the client name:

```
#ifdef COLOR
*customization:  -color
#endif
```

Note that you're not limited to using the customized files in the app-defaults directory. You can also use `#ifdef` conditionals with your own resource specifications. The following example introduces another level of the conditional (using `#else`) that allows you to specify resources that only apply on a monochrome screen.

```
#ifdef COLOR
! Place your own color resource specifications here
*Background:   whitesmoke
*Foreground:   darkorchid
xclock*background:  lightseagreen
xclock*foreground:  navy
#else
! Place your own monochrome settings here
xclock*reverseVideo:  True
#endif
```

Note that you can include your own specifications *and* custom color files in the same conditional:

```
#ifdef COLOR
! Use any customized color app-defaults files
*customization:  -color
! And your own definitions
*Background:   whitesmoke
*Foreground:   darkorchid
xclock*background:  lightseagreen
xclock*foreground:  navy
#endif
```

Or you can have multiple conditionals in the same *.Xresources* file:

```
! You might place your own settings in one conditional
#ifdef COLOR
*Background:   whitesmoke
*Foreground:   darkorchid
xclock*background:  lightseagreen
xclock*foreground:  navy
#endif

! And place any customized app-defaults files in another conditional
#ifdef COLOR
*customization:  -color
#else
*customization:  -mono
#endif
```

Note that the preceding example invokes app-defaults files ending in "–mono" when the screen is not color. Unless these files exist, the color defaults will be applied regardless of the screen and you may not be able to see all of the window's features.

Testing and Editing Resources with editres

The *editres* (*res*ource *edit*or) client is another welcome Release 5 innovation. By now you have an idea of the potential complexities surrounding resource settings and how they are intrepreted. *editres* is most useful in helping you to examine the often complicated hierarchy of a client's widgets and to devise correct resource specifications. Using *editres* you can:

- Display and scan through the client's widget hierarchy.
- Display what resources may be set for a particular widget.
- Create resource specifications.
- Dynamically apply the new specifications to a client already running on the display!
- Write the new definitions to your own resource file.

editres can be incredibly helpful, but it is not simple to use. In the following sections, we provide a tutorial that gives you an idea of what *editres* can do, but we will not cover every feature.

Note that the usefulness of the program will be limited somewhat by your understanding of widgets and the resources that can be applied to them. *editres* will show you *all* the resources that can be set for a widget, but it will not differentiate between those you can set at the user level and those that must be set by programming routines. Use the client reference pages in Part Three and Appendix G, *Widget Resources*, to get a better idea of what resources you can set yourself.

Note also that *editres* can only work with certain clients—those that understand the so-called *editres protocol*. Most clients built using the Athena widget set will work with *editres*. A Motif Toolkit client may not be compatible. If you try to use *editres* with an incompatible client, the following message will be displayed in the *editres* window:

```
It appears that this client does not understand
the Editres Protocol.
```

What Widget Is That, Anyway?

Let's consider a scenario in which *editres* would be helpful to a user. This case happens to be actual and we won't even change the names. One of my co-workers, Jerry, was trying to make screen dumps of the *xmh* client,* but he found that the default font being used for the menus was too small. He wanted to change the font for menu text only, leaving the default font for command buttons, etc. The specification:

```
xmh*font:  bigger_font
```

would change the font in *every* widget for which a font could be set. We used *editres* to determine the particular widgets for which to set the font.

In order for *editres* to examine a client's widget hierarchy, the client must be running, so we ran both *xmh* and *editres*:

```
% xmh &
% editres &
```

Figure 11-4 shows the clients.

xmh is the X version of the *mh* mail handler. Part Three of this guide includes a reference page for *xmh*. For further information, see the Nutshell Handbook *Using mh and xmh*, written by Jerry Peek and also published by O'Reilly & Associates, Inc.

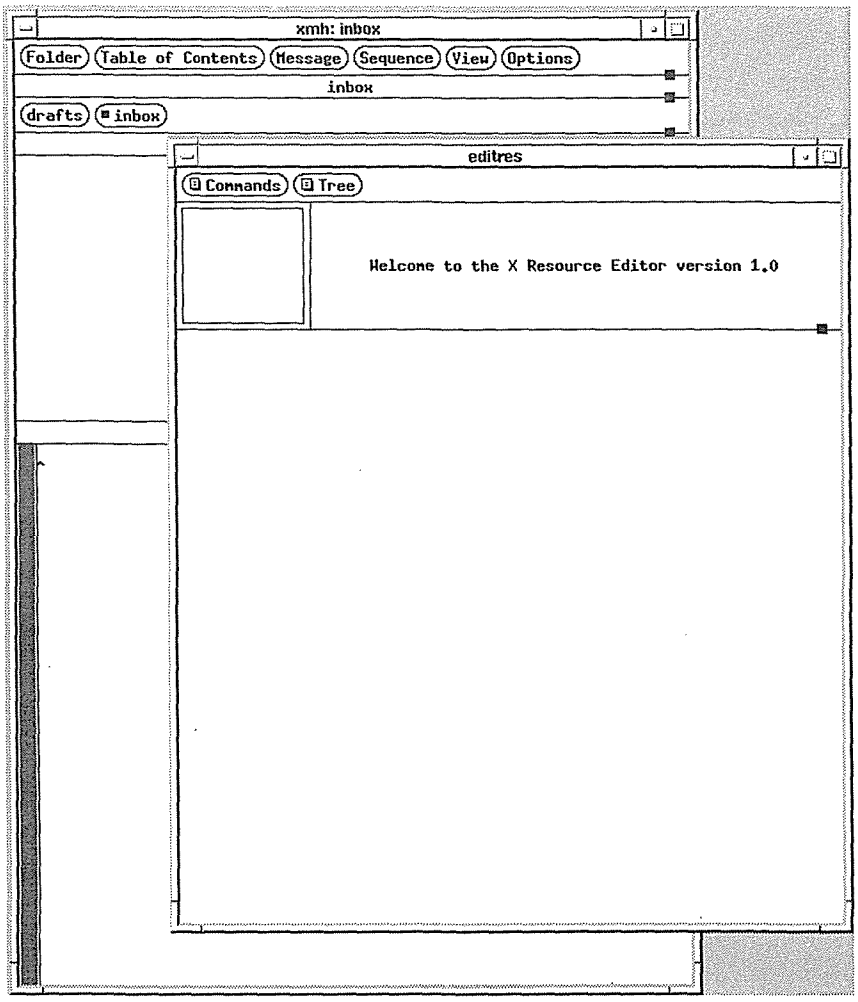

Figure 11-4. editres and xmh

editres Menus

editres provides two menus: Commands and Tree. To display a menu, place the pointer on the appropriate command button and press and hold down the first pointer button. Select an item by dragging the pointer down the menu and releasing on the item you want.

The most important things the Commands menu allows you to do are:

- Display a client's widget tree (Get Widget Tree);
- Access a subwindow (called the *resource box*) from which you can test, set, and save resource specifications (Show Resource Box);
- Quit *editres*.

The Tree menu helps you:

- Determine the correspondence between the widget tree and the actual widgets in the client (Select Active Widget; Flash Active Widgets).
- Select groups of widgets (parents, children, etc.) for subsequent operations (e.g., showing the widget in the actual client with Flash Active Widgets).

Displaying the Widget Tree

Now, how do we figure out what resource line to use to specify the font for *xmh* menus? First, we must display *xmh*'s widget tree. Select Get Widget Tree from the Commands menu and you will be prompted to

```
Click the mouse pointer on any Xaw client.
```

in the message window below and to the right of the menu buttons. Then click the cross pointer anywhere on the *xmh* window and the client's widget hierarchy is displayed in tree format in the *editres* window. *xmh* has a very complex hierarchy and only part of the tree can be viewed in the *editres* window, as shown in Figure 11-5.

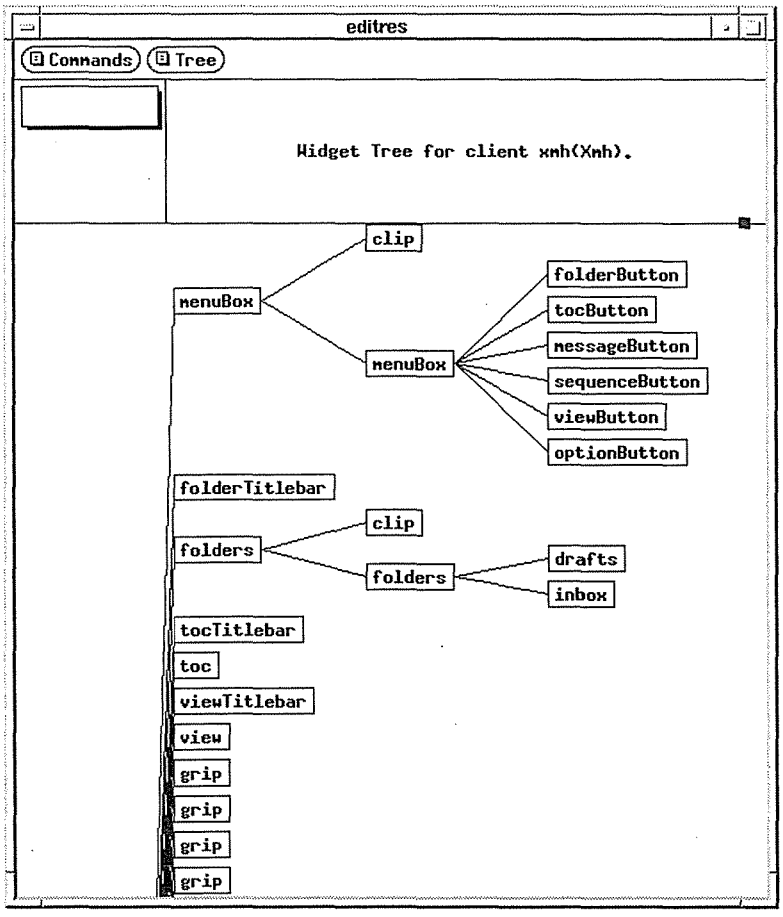

Figure 11-5. editres displays xmh's widget tree

Notice that the box beneath the menu command buttons has become smaller. This box is called the *panner* and it is actually a tool that allows you to scan the entire tree. The size and location of the panner in the larger square surrounding it suggests the portion of the widget tree that is visible—in this case, approximately the top third of the tree. To view the rest of the tree, place the pointer on the panner, hold the first pointer button and drag. The *editres* window scrolls to reveal the remaining widgets.

Tracking Down the Widgets

We're trying to set the font for *xmh* menu items like Open Folder on the Folder menu. Figure 11-5 displays a few widgets that sound as if they might be part of the menus. The `menuBox` widget and its children (to the right of it and connected to it by lines) seem promising. We can determine where the `menuBox` widget appears in the *xmh* application by performing the following actions:

1. Place the pointer on the `menuBox` square in the *editres* tree and click the first button. This action highlights the widget (which appears in reverse video); certain menu actions will affect only the highlighted widget(s).
2. Then select Flash Active Widgets from the Tree menu. The widget highlighed in the *editres* window will be "flashed" in the *xmh* window.

This action shows that the `menuBox` is the entire top portion of the *xmh* window—not even an individual menu. A look at the names of the widget's children, all of which end in `Button`, suggests that these are merely the menu command buttons and we have to examine the tree further to find the actual menu definitions.

Before scrolling, it's important to deselect the `menuBox` widget by clicking on it again. *editres* allows you to select multiple objects in order to perform certain operations, but we're only interested in one right now.

Figure 11-6 shows the *editres* window after we pan down to the more promising widget name `folderMenu`, the children of which seem to approximate this menu's choices.

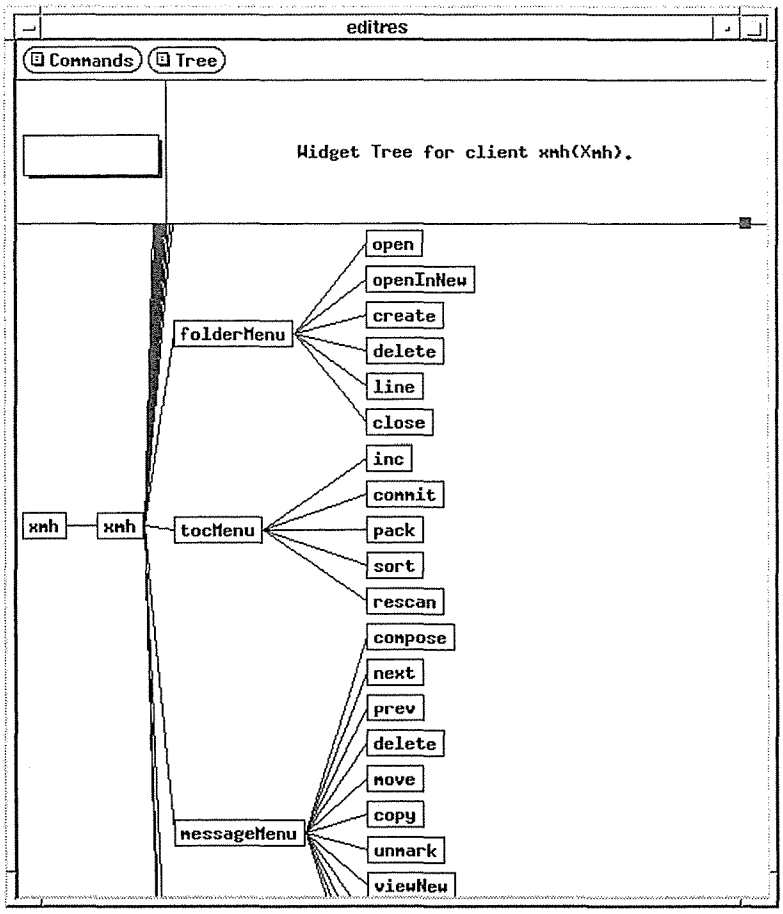

Figure 11-6. Middle portion of xmh's widget tree reveals menu items

Using the Resource Box to Create a Specification

Now we can try to write a resource to specify the font. First, select the `folderMenu` widget in the tree. Then select Show Resource Box from the Commands menu. The resource box subwindow appears on top of the main *editres* application window, as in Figure 11-7.

The resource box is fairly complicated, but we can determine one disappointing fact quickly. Neither list of resources it provides (Normal Resources and Constraint Resources) includes a font resource! Seems we're on the wrong track.

Click on the Popdown Resource Box button that appears in the lower righthand corner of the resource box. The box goes away. Now let's deselect `folderMenu` on the widget tree and instead try selecting one of its children, `open`, the widget of the first Folder menu item. Then select Show Resource Box from the Commands menu again. This time the resource box includes `font` under the list of Normal Resources (there are no constraint resources).

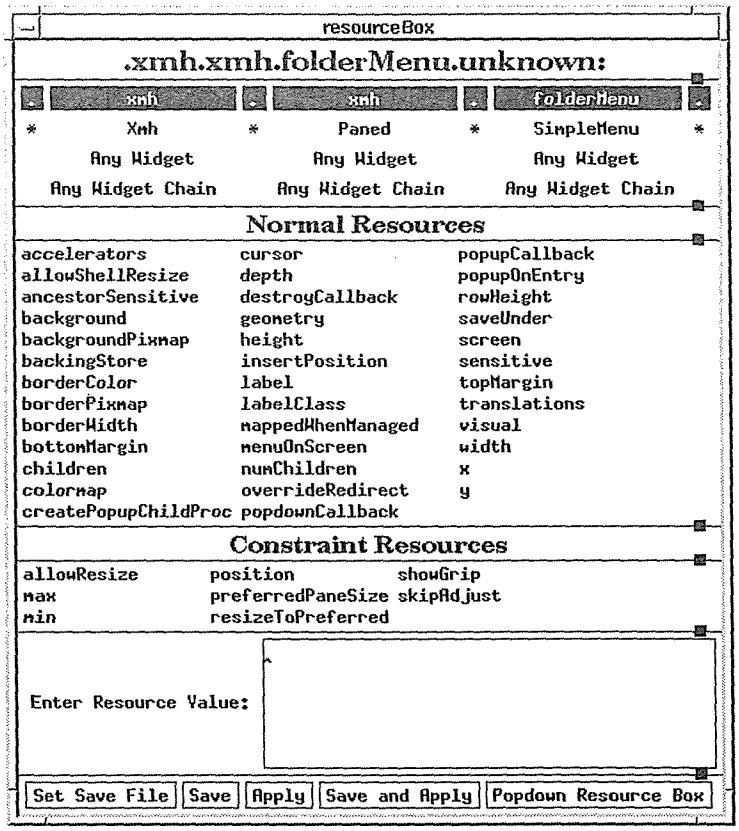

Figure 11-7. The editres resource box

Across the top of the resource box is a template resource specification for the selected widget, which at this stage shows the tightly bound instance name ending with an `unknown` resource variable. You can select the resource you want from the list in the box by highlighting it in the same way you did a widget in the tree—simply click the pointer on it. When we click on `font`, the `unknown` variable in the template is changed to `font`.

We still have to supply a value (a readable font), but first let's look at the specification more closely.

```
.xmh.xmh.folderMenu.open.font:
```

(Don't be confused by the two instances of `xmh` beginning the specification; the first represents the client and the second represents the next level widget in the client, which in this case has the same name! This is fairly unusual.) The `folderMenu` and `open` widgets suggest that this line would set the font for the Open Window item on the Folder Menu. But we need to specify a resource that will cover *all* menu items. At this point, you might be tempted to highlight the other menu item widgets in the tree, but if you do so and try to display the resource box again, you'll discover that the box can only be used when a single widget is highlighted.

Instead, *editres* provides a way for you to edit the template resource specification. Notice that below the template are four lines of text, the first one of which matches the full instance name in the template, with each component (including connectors) highlighted. The next

line down shows the full class name with loose bindings. (We'll discuss the third and fourth lines in the next section.) As you can see, the four lines are spaced so that the components and connectors fall into columns. These lines provide four sets of alternatives for each of the components in the template resource specification. As you move the pointer around among the various choices, notice that a box highlights each one in turn. You can change any part of the template specification by clicking on an alternative in the same column.

For instance, to switch any tight binding in the template to a loose binding, you would simply click on the corresponding loose binding on the class name line. The highlighting for that column will be switched to loose binding on the class name line and the template will be redrawn to include the asterisk.

You can also replace any component in the template by clicking on the alternative component you want. Since we want to specify a font resource that applies to all menus, perhaps we should select the class name that corresponds to the `folderMenu` widget, `SimpleMenu`. When we click the pointer on `SimpleMenu`, that class name is highlighted and replaces `folderMenu` in the template line (`folderMenu` is also unhighlighted), as in Figure 11-8.

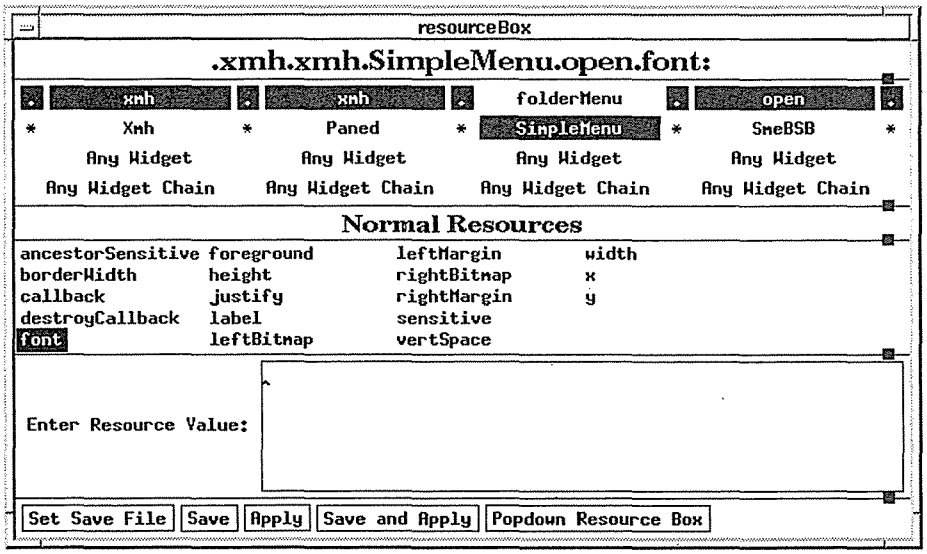

Figure 11-8. Edit the template resource by clicking on another component

Now we're close to the solution, but the `open` widget is still too specific. We then click on the corresponding class name, `SmeBSB` to produce the template line:

```
.xmh.xmh.SimpleMenu.SmeBSB.font:
```

If we move the resource box away from the main *editres* window, we'll see that all of the widgets covered by this specification (i.e., those corresponding to the menu items) are now highlighted.

Now we must enter a value for this resource (a font name) in the text window near the lower right corner of the *editres* window. The phrase "Enter Resource Value:" appears to the left of the text window. To enter a value, place the pointer in the text window and type. (The text

window is an instance of the Athena Text widget. To learn the valid editing commands, see "The xedit Text Editor" in Chapter 8, *Other Clients*.) We enter a fixed width font with the alias *7x14*.

Here's where *editres* comes in very handy. We can test our specification on the currently running *xmh* client. Just click on Apply, the third of five command buttons that appear along the bottom of the resource box. If the template resource can be applied successfully to the client in question, the message area to the right of the panner will display:

```
SetValues was Successful.
```

We verify that our new font resource has been incorporated into the running client by displaying one of the *xmh* menus and *7x14* seems large enough to reproduce well in the screen dump Jerry wants to make. Figure 11-9 shows one of the resulting screen dumps, borrowed from Jerry's book, *Using mh and xmh*.

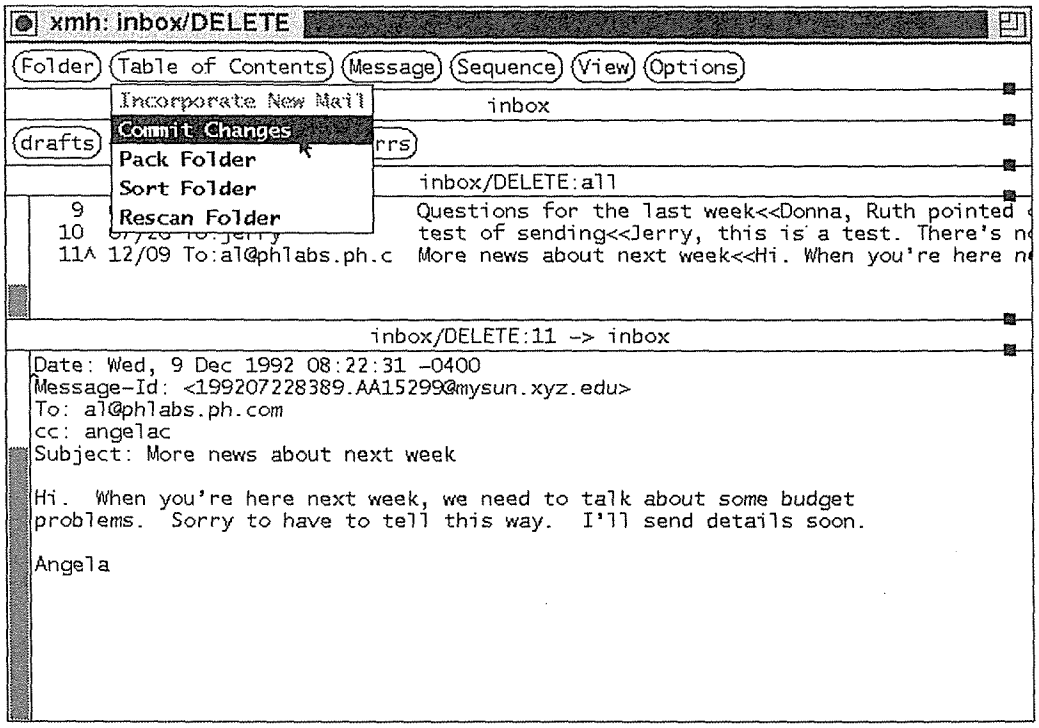

Figure 11-9. Custom font makes xmh menus readable in screen dump

All of the command buttons at the bottom of the resource box help you either "Apply" the custom resource specification to the running client or "Save" the specification in a resource file. Set Save File prompts you to specify the resource filename:

```
Enter file to dump resources into:
```

If you select any command with "Save" in it and haven't previously provided the filename, you will be prompted in the same way before the command proceeds.

Other Ways to Specify the Same Resource

Our sample resource line:

```
.xmh.xmh.SimpleMenu.SmeBSB.font:  7x14
```

will do for our purposes, but the following resources would accomplish the same result:

```
.xmh.xmh.?.SmeBSB.font:  7x14
.xmh.xmh*SmeBSB.font:  7x14
```

These resource specifications illustrate the use of the alternative components in the third and fourth lines below the template in the resource box. If you look back at Figure 10-5, you'll see that these lines offer the phrases Any Widget and Any Widget Chain as alternatives at every level in the resource specification.

Selecting Any Widget replaces the component with the question mark wildcard, which represents exactly one widget level. (See "Wildcarding a Component Name with ?" earlier in this chapter.) If you go back to our example in the last section and select Any Widget in the third column, the question mark replaces the menu widget in the template:

```
.xmh.xmh.?.SmeBSB.font:  7x14
```

If you apply this resource specification, you'll find it accomplishes the same thing our complete specification does.

Selecting Any Widget Chain removes the component from the template line and elides the surrounding components using an asterisk wildcard (loose binding). (See "Tight Bindings and Loose Bindings" earlier in this chapter.) Again, go back to our example in the last section and this time select Any Widget Chain in the third column. An asterisk replaces the menu widget in the template and the surrounding tight bindings are removed.

```
.xmh.xmh*SmeBSB.font:  7x14
```

Again, this resource accomplishes the same thing.

Of course, there are several other alternative resource specifications you could also use. We won't get into them here. The most important thing to remember is that *editres* (and the resource manager itself) allow you to do a bit of experimenting.

12

Specifying Color

This chapter gives an overview of the color names you can use in virtually any X environment. These names specify so-called RGB colors, which look different on different monitors. Release 5 introduces support for device-independent color. This chapter explains the principles behind these two "color models" and describes how to specify colors according to each model. It also explains how to create your own color database.

In This Chapter:

12
Specifying Color

What Color Names Can I Use?

As mentioned in Chapter 10, *Command-line Options*, specifying colors can become rather complicated, but you can also keep it simple. The next section ("Available RGB Colors") gives an overview of the basic color names you should be able to use in virtually any environment—without any system customization. If you think the set of predefined colors will be sufficient for your needs, you shouldn't have to read more than that section (and its subsections).

If you want to know more about color (including how to create a private database of your own colors), you'll need a little more background. X actually recognizes two "color models": a server-side (non-portable) RGB color model (primarily a database) and a client-side (device-independent) X Color Management System (Xcms). Keep in mind that you can use take advantage of both color models. (See "A Mixed Bag: Using both RGB and Xcms" for an explanation of precedence.) Xcms has been introduced in Release 5; if you are satisfied with the older RGB functionality, you can ignore Xcms. However, it does have many interesting and powerful capabilities. In the remainder of this section, we'll give you an overview of both of these models.

The RGB color model includes a database of color names which can be accessed by the server. This database exists in a readable form in a system file called *rgb.txt*, which generally lives in the directory */usr/lib/X11*. (Most of the color names you can think of—plus many esoteric names—are included in this database.) Within the *rgb.txt* file, each color name is associated with three numeric values, corresponding to the amount of red, green, and blue in the shade. The display hardware uses these values to produce a color. (For a more technical discussion of this color model, see "Beyond the Rainbow: Inside the Color Models" later in this chapter.)

The RGB model accepts color values in two different formats: color names from the database (which pairs names with decimal values for red, green, and blue); numeric color values in hexadecimal notation. (You can use either of these on command lines, in resource files, etc.) As we'll see later in this chapter, the hex values allow you to specify a wider spectrum of colors than is covered by the database.

From a user's standpoint, the advantages of the RGB database are:

- It provides a wide range of colors whose names you can simply plug in to command lines and resource specifications;
- It requires no customization, but allows customization if you're ambitious enough to want to come up with your own colors or tune existing ones. (Note, however, that since it is available on a system-wide basis, the RGB database is generally the private preserve of the system administrator.)

The primary disadvantage of the RGB database (and the model itself) is that the colors it defines can look very different on different types of display hardware. This is because the server accesses the database (or the numeric hex specifications) and simply applies the color values, without any tuning for the type of monitor, the platform, etc. Certain intensities of red, green, and blue might produce orange on one display and pink on another. In other words, RGB colors are hardware-specific and thus, non-portable.

We speculate that most users won't care too much if their "pink" is too "orangy" and will just experiment with other colors in the *rgb.txt* file to find shades they like. (The public domain *xcol* program, described in Chapter 8, will help you preview the *rgb.txt* colors for your monitor.)

If you are interested in coming up with your own colors, there are several public-domain "color editors" available. See Chapter 8, *Other Clients*, for instructions on using two of these clients, *xcoloredit* and *xtici*. Once you come up with your own shade, the RGB model allows you to either add the color name/value pair to the database (see "Changing the RGB Color Name Database" later in this chapter); or to specify the alternative color on the command line (or in a resource file) by providing its value in hexadecimal notation (see "Specifying RGB Colors as Hexadecimal Numbers"). Keep in mind, however, that a color in RGB format is *not* portable. If you use a variety of monitors, an RGB color (even one you create with a color editor) will look different on each of them.

If you're not entirely satisfied with the hardware-specific RGB model, an additional, more precise color model has been made available in Release 5. The X Color Management System (Xcms), developed by Tektronix (and now adopted by the X Consortium), is intended to overcome the limitations of the (still available) RGB model by providing *device-independent* color. Under Xcms, colors are based upon internationally recognized standards. The idea behind Xcms is that color relies upon human vision. Simply put, red should look basically the same on any monitor, under any platform.

Xcms accepts color values in several different formats, called *color spaces* (outlined later in this chapter). Most of these color spaces describe color in a device-independent manner, using scientific terms and values commonly applied to color. (For more information, see "The Xcms Color Spaces" later in this chapter.) Note, however, that Xcms will recognize an RGB color space—only the values are not portable.

From a user's standpoint, the primary advantages of Xcms are:

- It recognizes several types of color specification (color spaces), which can be supplied on the command line or in resource files.

- It enables you to make a database of colors you "mix" yourself, using a color editor. (While the server-side RGB model allows for a single system-wide database, Xcms allows any user to have a private database.) In the Xcms database, you pair a name with a value in any of the accepted color spaces (formats). The Xcms database can then serve as an alternative to the default RGB database (you can specify colors from either).
- Xcms can "tune" colors to display more accurately on specific hardware. (Some X terminals are configured to do this automatically; on most other displays, you need to install a special file to perform this tuning. For more information, see "The X Color Management System" later in this chapter.)
- Xcms allows users to take advantage of sophisticated color printer technology.

If you're at all interested in coming up with your own colors, it's worth reading more about Xcms (later in this chapter) and learning how to create your own database. Your system administrator might also want to create a system-wide Xcms database. (See "Creating an Xcms Color Database" later in this chapter for details.)

But before we go any deeper into the technology of color, let's take a look at the simplest source of color specifications: the RGB database. Then we'll consider the Xcms color spaces (formats), which have been made available in Release 5. Finally, we'll take a more technical look behind the color models and learn how to make additional color names available under either model. If you want to customize the RGB database or possibly create an Xcms database of your own, read the relevant sections.

Available RGB Colors

As previously mentioned, the server-side RGB color model allows you to specify colors using either:

- Text names from the standard database (*rgb.txt* file);
- Numeric color values in hexadecimal notation.

The simplest way to specify colors for your display is to look over the *rgb.txt* file and pick some names. The many colors defined in this file are probably more than most users will need.

Specifying a color as a hexadecimal number comes in handy when you've used a color editor, such as *xcoloredit*, to create a new shade and the program gives its output in hex. Chapter 8, *Other Clients*, describes how to use *xcoloredit*. The section "Changing the RGB Color Name Database" later in this chapter describes how to add one of these new colors to the *rgb.txt* file so that you can access it by a name.

The following sections give you an idea what colors are available in the RGB database. Then we'll take a look at specifying a color as a hex value.

Surveying the RGB Database: rgb.txt

The *rgb.txt* file, usually located in */usr/lib/X11*, is supplied with the standard distribution of X and consists of predefined color values (in decimal notation) assigned to specific text names. Corresponding compiled files called *rgb.dir* and *rgb.pag* contain the definitions used by the server; these machine-readable files serves as a color name database. The *rgb.txt* file is the human-readable equivalent.

The default *rgb.txt* file shipped with Release 5 of X contains 738 color name definitions. This number is slightly deceptive, since a number of the color names are merely variants of another color name (differing only in spelling, spacing, and capitalization). Still others are shades of the same color:

```
light sea green
sea green
medium sea green
dark sea green
SeaGreen1
SeaGreen2
SeaGreen3
SeaGreen4
DarkSeaGreen1
DarkSeaGreen2
DarkSeaGreen3
DarkSeaGreen4
```

Each of these names corresponds to a color definition (of three RGB values). (This list does not include the variant syntax names SeaGreen, LightSeaGreen, MediumSeaGreen, and DarkSeaGreen, which also appear in the file.) As you can see, some of these shades are distinguished in the fairly traditional way of being called "light," "medium," and "dark." The light, medium, and dark shades of a color can probably be distinguished from one another on virtually any monitor.

Beyond this distinction, there are what might be termed "sub-shades": gradations of a particular shade identified by number (SeaGreen1, SeaGreen2, etc.). Numerically adjacent subshades of a color may not be clearly distinguishable on all monitors. For example, SeaGreen1 and 2 may look very much the same. (You certainly would not choose to create a window with a SeaGreen1 background and SeaGreen2 foreground! On the other hand, subshades a couple of numbers apart are probably sufficiently different to be used on the same window.)

By supplying many different shades of a single, already fairly precise color like sea green, X developers have tried to provide definitions that work well on a variety of commonly used monitors.* You may have to experiment to determine which colors (or shades) display best on your monitor. (For *device-independent* color, use the Xcms color model.)

*The color database shipped with prior releases of X was originally designed to display optimally on the vt240 series terminals manufactured by Digital Equipment Corporation.

The color names in the *rgb.txt* file are too numerous to list here. Although there are no literal dividers within the file, it can roughly be considered to fall into three sections:

Section 1: A standard spectrum of colors (red, yellow, sea green, powder blue, hot pink, etc.), which seem to be ordered roughly as: off-whites and other pale colors, grays, blues, greens, yellows, browns, oranges, pinks, reds, and purples.

Section 2: Sub-shades of Section 1 colors (such as SeaGreen 1 through 4). These sub-shades make up the largest part of the file.

Section 3: One hundred and one additional shades of gray, numbered 0 through 100. This large number of precisely graduated grays provides a wide variety of shading for grayscale displays.

Rather than list every color in the *rgb.txt* file, we've compiled this table of representative colors. We've chosen some of the more esoteric color names. Naturally all of the primary and secondary colors are also available.

Section 1:

ghost white	peach puff	lavender blush	lemon chiffon
slate gray	midnight blue	cornflower blue	medium slate blue
dodger blue	powder blue	turquoise	pale green
lawn green	chartreuse	olive drab	lime green
khaki	light yellow	goldenrod	peru
sienna	sandy brown	salmon	coral
tomato	hot pink	maroon	violet red
magenta	medium orchid	blue violet	purple

Section 2:

snow1 - 4	bisque1 - 4	cornsilk1 - 4	honeydew1 -4
azure1 - 4	SteelBlue1 - 4	DeepSkyBlue1 - 4	LightCyan1 - 4
PaleTurquoise1 - 4	aquamarine1 - 4	PaleGreen1 - 4	DarkOliveGreen1 - 4
SpringGreen1 -4	gold1 - 4	RosyBrown1 - 4	burlywood1 - 4
chocolate1 - 4	firebrick1 - 4	DarkOrange1 - 4	OrangeRed1 - 4
DeepPink1 - 4	PaleVioletRed1 - 4	plum1 - 4	DarkOrchid1 - 4

Section 3:

gray0 (grey0) through gray100 (grey100)

If you want to look more closely at the *rgb.txt* file, you can open it with any text editor. As an alternative, you can display the contents of the file using the *showrgb* client. *showrgb* seems to do nothing more than *cat*(1) the file to your terminal window. In fact, it consults the database (*dbm*) version of the file. Given the size of the database, it's necessary to pipe the command's output to a paging program, such as *pg*(1) or *more*(1), as shown below:

```
% showrgb | more
```

Be aware that *showrgb* will display the color names in a different order than they appear in *rgb.txt*. See "Changing the RGB Color Name Database" for information on customizing the RGB color name definitions.

Keep in mind that RGB colors look different on different monitors. (Later in this chapter, we'll take a closer look at the Xcms Color Management System, which provides device-independent color capabilities.) The *xcol* client, from the user-contributed distribution, allows

you to display the colors defined in the *rgb.txt* file. *xcol* can also be used to edit the color specifications in a resource file. See Chapter 8, *Other Clients*, and the *xcol* client reference page in Part Three of this guide.

Alternative Release 5 RGB Color Databases

In addition to the standard color database described previously, Release 5 also includes three other databases your system administrator can compile. These files can be found in the general release in the directory *mit/rgb/others*.

raveling.txt	Designed by Paul Raveling, this database rivals the default database in size and scope but was tuned to display optimally on Hewlett-Packard monitors.
thomas.txt	Based on the Release 3 database, this file has been modified by John Thomas of Tektronix to approximate the colors in a box of Crayola Crayons.
old-rgb.txt	This is nothing more than the Release 3 database.

Specifying RGB Colors as Hexadecimal Numbers

Each RGB color has a numeric value associated with it. In the *rgb.txt* file, these values are in decimal notation—the base 10 numbering system with which most of us are familiar—and they are paired with a color name. But the RGB model also allows you to specify a color using *only* a number—in an alternative format known as hexadecimal notation. Hex is a base 16 numbering system used in many scientific disciplines. We'll discuss the system in more depth in "The RGB Color Model" later in this chapter.

Being able to specify a color by a numeric value means that you can be very precise. It also means that you can define virtually an infinite number of shades—certainly far more than is practical to define in a database like *rgb.txt*. Of course, it's also only practical to *use* a certain number of colors and hardware provides its own limitations. (See "How Many Colors Are Available?" for a clearer idea.)

Being able to supply hex values comes in very handy if you're using a color editor, such as *xcoloredit* (described in Chapter 8). When you "mix" your own color using *xcoloredit*, the program outputs the numeric color value in hexadecimal notation. For example, in Chapter 8 we came up with a bright shade of blue with the hex value:

```
09e5fb
```

We can then supply this number on the command line or in a resource file, prefixed with a pound sign (#), as in the following example:

```
xbiff -fg #09e5fb &
```

On our Sun monitor (remember, RGB values are hardware-specific), this command line produces an *xbiff* window with a bright blue mailbox. The following line from a resource file would have the same effect.

```
xbiff*foreground:  #09e5fb
```

See "Changing the RGB Color Name Database" for instructions on converting hex to decimal and pairing these values with names in the *rgb.txt* file. For more about the RGB model and how hex numbering works, see "The RGB Color Model" later in this chapter.

The Xcms Color Spaces

As of Release 5, you are not limited to supplying colors from the *rgb.txt* file or providing hex equivalents of RGB colors. The X Color Management System recognizes color specifications in many different formats called *color spaces*. Most of these formats reflect international standards. (Xcms also recognizes RGB decimal values, though these remain device-specific). This section gives an overview of the valid color spaces.

As with the server-side RGB model, you can supply the Xcms color spaces on the command line and in resource files. You can also create a custom color database. (Note that the standard colors from *rgb.txt* will still be available.) See "Creating an Xcms Color Database" for instructions.

Under Xcms, each color specification has a prefix (some shorthand for the color space) and a numeric value. Table 12-1 summarizes the valid color spaces and their prefixes.

Table 12-1. Xcms Color Spaces

Name	Prefix
Tektronix HVC	TekHVC
various CIE formats	CIEXYZ, CIEuvY, CIExyY, CIELab, CIELuv
RGB	RGB
RGB Intensity	RGBi

Tektronix is the developer of the X Color Management System. The initials HVC refer to hue, value, and chroma, scientific characteristics of color. (See "Creating Your Own Colors: xcoloredit and xtici" in Chapter 8, *Other Clients*, for more information.) CIE stands for *Commission Internationale de l'Eclairage* or *International Commission on Illumination*, an international standards organization.

Of the valid color spaces, the Tektronix HVC and the various CIE formats specify color in a device-independent manner. The RGB color spaces specify color that is hardware-specific. To take advantage of the portability of the X Color Management System, you will want to

use TekHVC or any of the CIE formats. Xcms recognizes RGB specifications for compatibility with the older RGB color model.

When you create a shade with a color editor, such as *xcoloredit* or *xtici*, the program supplies you with a numeric color value in one or sometimes multiple formats. To specify the color under Xcms, you combine the numeric value with the appropriate prefix for the color space/format. The syntax is:

```
prefix:value1/value2/value3
```

The following are sample Xcms color specifications:

```
CIEuvY:0.15259/0.40507/0.44336
TekHVC:223.93036/72.45283/29.67013
RGB:6a/bb/d8
```

These three sample values were derived from *xtici* (being run on a Sun 3/60 workstation); the values all define the deeper version of sky blue from the tutorial in Chapter 8, each using a different notation.* Keep in mind that the RGB value is specific to the monitor used, while the TekHVC and CIEuvY values are portable. We can supply *any* of these color spaces on our Sun 3/60 and get the same color. Thus, the following command lines should produce identical *xbiff* windows:

```
xbiff -fg CIEuvY:0.15259/0.40507/0.44336 &
xbiff -fg TekHVC:223.93036/72.45283/29.67013 &
xbiff -fg RGB:6a/bb/d8 &
```

(Note that the Xcms color spaces are case insensitive. Thus, `rgb:6a/bb/d8` and `RGB:6A/BB/D8` are equivalent.) If we want to display this shade of blue on another monitor, we would have to use either of the portable specifications:

```
CIEuvY:0.15259/0.40507/0.44336 &
TekHVC:223.93036/72.45283/29.67013 &
```

You can also use any valid color space as the value of a resource variable:

```
xbiff*foreground: TekHVC:223.93036/72.45283/29.67013
```

It's handy to be able to plug these numbers into a command line or resource specification, but if you want to use your own colors on a regular basis, it's a good idea to pair them with names in your own Xcms database. First read a bit more about Xcms later in this chapter. Then see "Creating an Xcms Color Database" for instructions.

**xtici* outputs each of the three color spaces (CIEuvY, TekHVC, and RGB) in a format Xcms understands, but handles RGB values in a somewhat confusing manner. It accepts input (and displays the RGB values) in decimal notation, but outputs RGB hex values (when you request the RGB values via the Edit menu). You can use decimal numbers in the RGB server-side color database (*rgb.txt* file; see "Changing the RGB Color Name Database"); however, you should use hexadecimal notation for Xcms to recognize the values on the command line, in a resource file, or in an Xcms database (described in "Creating an Xcms Color Database"). "Finding the Color Values" provides instructions on performing this conversion exclusive of the color editor—using the UNIX *bc*(1) utility.

A Mixed Bag: Using Both RGB and Xcms

All this talk of color models can get pretty confusing. But take heart. Even if you use both, in practice you shouldn't have to think too much about it. You can supply color specifications in any form acceptable to either color model and X will resolve any possible conflicts.

X searches to match a color specification in this order:

1. If it begins with the pound sign (#), the subsequent number is interpreted as a hexadecimal RGB value.
2. If it contains a colon (:), the prefix is checked to see if it matches a valid color space; if it does, the subsequent number is interpreted as a value in that color space. All currently valid color spaces recognize the forward slash (/) as the delimiter between numeric values. (Each color space defines its own delimeter, so hypothetically new formats may recognize other delimeters.)
3. If the color specification contains neither a pound sign or a colon, it is assumed to be a color name that should appear in either an Xcms database or the RGB server database. (The Xcms database is checked first, so if a color name appears in both databases, the Xcms color value is applied.)

Beyond the Rainbow: Inside the Color Models

The following sections take a more technical look at the two color models. If you want to edit the RGB database or create an Xcms database of your own, this information might be helpful. Keep in mind, however, that the X Color Management System is far beyond the scope of this guide. For some additional information, see Volume Eight, *X Window System Administrator's Guide*.

The RGB Color Model

Most color displays on the market today are based on the RGB color model. Each pixel on the screen is actually made up of three phosphors: one red, one green, and one blue. Each of these three phosphors is illuminated by a separate electron beam, called a *color gun*. These color guns can be lit to different intensities to produce different colors on the screen.

When all three phosphors are fully illuminated, the pixel appears white to the human eye. When all three are dark, the pixel appears black. When the illumination of each primary color varies, the three phosphors generate a subtractive color. For example, equal portions of red and green, with no admixture of blue, makes yellow.

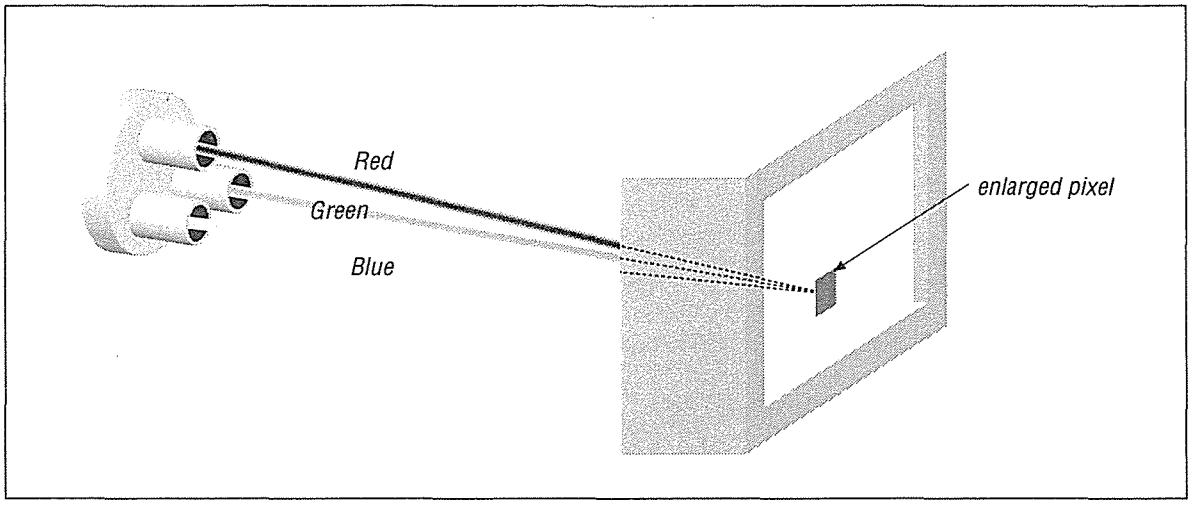

Figure 12-1. Red, green, and blue color guns

As you might guess, the intensity of each primary color is controlled by a numeric value. In the *rgb.txt* file, each color is associated with a decimal number between 0 and 255. Consider the following line from the *rgb.txt* file:

```
173 216 230             light blue
```

The three numbers make up what is known as an *RGB triplet*. As we've seen, the *rgb.txt* file contains 738 mappings of RGB triplets to color names. Inherent in this system is a limitation: a color name is associated with hard-coded intensities of red, green, and blue; these intensities may look different on different displays and under different implementations of the X server. (The X Color Management System transcends these limitations.)

An RGB triplet can also be supplied in hexadecimal notation, which can afford greater precision than the decimal values in *rgb.txt*. Depending on the underlying hardware, different servers may use a larger or smaller number of bits (from 4 to 16) to describe the intensity of each primary. To insulate you from this variation, most clients are designed to take color values containing anywhere from 4 to 16 bits (1 to 4 hex digits), and the server then scales them to the hardware. As a result, you can specify hexadecimal values in any one of these formats:

```
#RGB
#RRGGBB
#RRRGGGBBB
#RRRRGGGGBBBB
```

where R, G, and B represent single hexadecimal digits and determine the intensity of the red, green, and blue primaries that make up each color.

When fewer than four digits are used, they represent the most significant bits of the value. For example, `#3a6` is the same as `#3000a0006000`.*

*If you are unfamiliar with hexadecimal numbering, see the Glossary for a brief explanation or a basic computer textbook for a more extended discussion.

What this means concretely is perhaps best illustrated by looking at the values that correspond to some colors in the color name database. We'll use 8-bit values—two hexadecimal digits for each primary. These definitions are the hexadecimal equivalents of the decimal values for some of the colors found in the *rgb.txt* file:

```
#000000     black
#FFFFFF     white
#FF0000     red
#00FF00     green
#0000FF     blue
#FFFF00     yellow
#00FFFF     cyan
#FF00FF     magenta
#5F9EA0     cadet blue
#6495ED     cornflower blue
#ADD8E6     light blue
#B0C4DE     light steel blue
#0000CD     medium blue
#000080     navy blue
#87CEED     sky blue
#6A5ACE     slate blue
#4682B4     steel blue
```

As you can see from the colors previously given, pure red, green, and blue result from the corresponding bits being turned on fully. Turning all primaries off yields black, while turning all nearly full on produces white. Yellow, cyan, and magenta can be created by pairing two of the other primaries at full intensity. The various shades of blue shown previously are created by varying the intensity of each primary—sometimes in unexpected ways.

Of course, fiddling with the numbers is fairly unintuitive. If you want to play with color, use a color editor. *xcoloredit* supplies its output in hex. See Chapter 8, *Other Clients*, for instructions.

The X Color Management System

As previously described, the X Color Management System is intended to provide device-independent color. There are actually two components to the system:

- The color spaces or formats (which should look the same on any system);
- *Device Color Characterization* (DCC) data that "tunes" color specifications for a particular hardware display.

We've already taken a look at some of the valid color spaces. If you specify a color using one of the portable Xcms color spaces, you should get the same color regardless of the monitor, server, etc.

Application developers may choose to fine tune Xcms colors for a particular hardware display by additionally installing a *Device Color Characterization* (*DCC*) file (also called a *Device Profile*). This tuning is an optional part of the system. You would probably need two adjacent monitors to see the difference between a system accessing DCC data and one not. The color spaces alone should be sufficient for most users. If you are interested in installing

DCC data, the section "Device-Specific tuning" offers some tips. See Volume Eight, *X Window System Administrator's Guide*, for more information.

How Many Colors Are Available on My Screen?

Regardless of the "model" you use to specify colors, the number of distinct colors available on the screen at any one time depends on the amount of memory available for color specification. (The *xdpyinfo* client provides information about a display, including the number of colors available at one time. See Chapter 8, *Other Clients*, and the *xdpyinfo* reference page in Part Three of this guide for details.)

A color display uses multiple bits per pixel (also referred to as multiple planes or the *depth* of the display) to select colors. Programs that draw in color use the value of these bits as a pointer to a lookup table called a *colormap*, in which each entry (or *colorcell*) contains the RGB values for a particular color.* (Xcms translates its device-independent values into the equivalent RGB values appropriate for the particular monitor.) As shown in Figure 12-2, any given pixel value is used as an index into this table—for example, a pixel value of 16 will select the 16th colorcell.

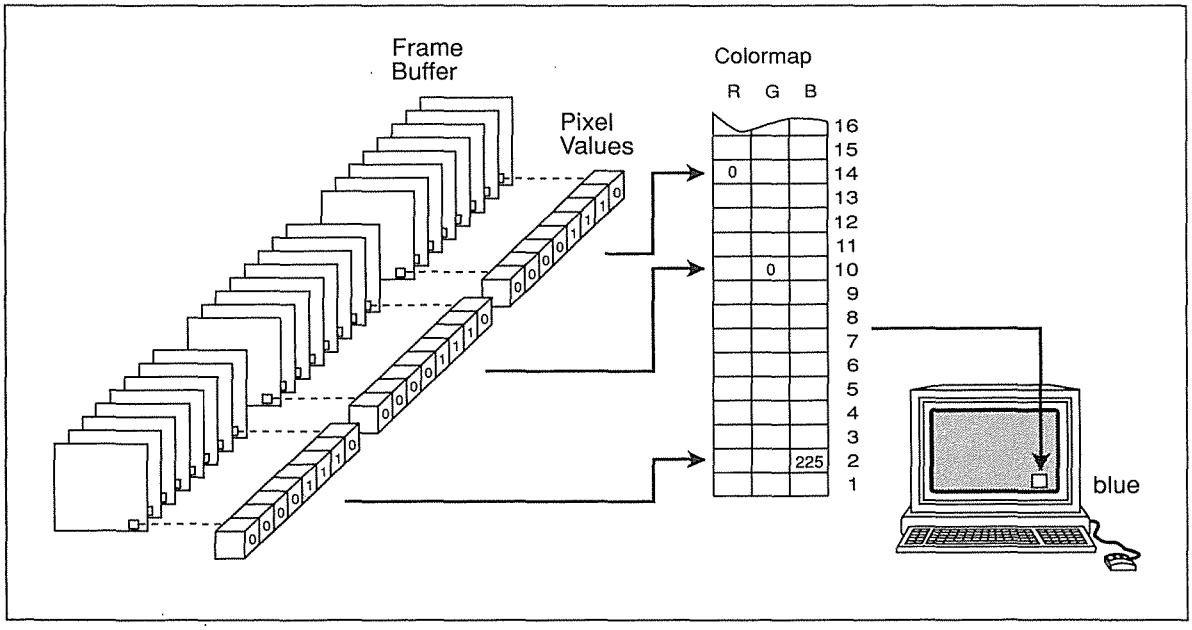

Figure 12-2. Multiple planes used to index a colormap

Why is this technical detail important? Because it explains several issues that you might encounter in working with color displays.

*There is a type of high-end display in which pixel values are used directly to control the illumination of the red, green, and blue phosphors. But far more commonly the bits per pixel are used indirectly with the actual color values specified independently.

First, the range of colors possible on the display is a function of the number of bits available in the colormap for RGB specification. If 8 bits are available for each primary, then the range of possible colors is 256^3 (more than 16 million colors). This means that you can create incredibly precise differences between colors.

However, the number of different colors that can be displayed on the screen at any one time is a function of the number of planes. A four-plane system can index 2^4 colorcells (16 distinct colors); an 8-plane system can index 2^8 colorcells (256 distinct colors); and a 24-plane system can index 2^{24} colorcells (more than 16 million distinct colors).

If you are using a 4-plane workstation, the fact that you can precisely define hundreds of different shades of blue is far less significant than the fact that you can't use them all at the same time. There isn't space for all of them to be stored in the colormap at one time or any mechanism for them to be selected even if they could be stored.

This limitation is made more significant by the fact that X is a multi-client environment. When X starts up, usually no colors are loaded into the colormap. As clients are invoked, certain of these cells are allocated. But when all of the free colorcells are used up, it is no longer possible to request new colors. When this happens, you will usually be given the closest possible color from those already allocated. However, you may instead be given an error message and told that there are no free colorcells.

In order to minimize the chance of running out of colorcells, many programs use *shared* colorcells. Shared colorcells can be used by any number of applications but they can't be changed by any of them. They can only be deallocated by each application that uses them, and when all applications have deallocated the cell, it is available for setting one again. Shared cells are most often used for background, border, and cursor colors.

Alternately, some clients have to be able to change the color of graphics they have already drawn. This requires another kind of cell, called *private*, which can't be shared. A typical use of a private cell would be for the palette of a color-mixing application, such as *xcoloredit*. This program has three bars of each primary color and a box that shows the mixed color. The primary bars use shared cells, while the mixed color box uses a private cell.

In summary, some programs define colorcells to be read-only and sharable, while others define colorcells to be read/write and private.

To top it off, there are even clients that may temporarily swap in a private colormap of their own. If this happens, all other applications will be displayed in unexpected colors because of the way color is implemented.

In order to minimize such conflicts, you should request unique numerical colors only when necessary. By preference, use color names or numerical specifications that you've given for other applications.

Adding New Color Names

Both the server-side RGB color model and the X Color Management System allow you to specify a color as a numeric value. Since it's much simpler to use a color name, both models also allow for a database in which numeric values are paired with names.

We've already considered the advantages and limitations of the RGB and Xcms color models. Most importantly, Xcms offers the advantage of portability: the color specifications are intended to be device-independent (i.e., they should look the same on any display hardware).

When you come up with your own color using a color editing program, such as *xtici* or *xcoloredit*, you'll know how that shade is going to look on your screen. If the value you come up with is a one of the portable Xcms color spaces, the color should look the same on any screen. If you intend to use that color multiple times, you'll probably want to pair it with a name in a database. The following sections describe how to edit the default RGB database or to create an Xcms database.

Although creating a color database can be very useful, keep in mind one less than obvious limitation: no color database can be accessed across the network. In other words, if you create an Xcms color database on your own workstation, you can't use any of those color definitions in a remote process. This limitation has nothing to do with whether a particular color specification is portable. A portable color should look the same on any system—but you have to specify the color on that system or have access to the color definition in a database local to the client.

Changing the RGB Color Name Database

The X Window System comes with a predefined set of colors, listed in the file */usr/lib/X11/rgb.txt*. As we've seen, you can use these color names to specify colors either on the command line or in a resource file. If you have access to a color editing program, such as *xtici* or *xcoloredit*, you can also come up with your own colors and add them to the RGB database.

Finding the Color Values

Each color in the RGB database has a name and a three-component numeric value, in decimal form. The fact that *xtici* displays the RGB values in the necessary decimal form is at least one advantage to using this program to come up with your own color(s). (See Chapter 8, *Other Clients*, for instructions on using *xtici*.) If you have a color value consisting of three decimal numbers, you can skip to the next section "Editing and Compiling the RGB Database."

xcoloredit outputs the color value in hexadecimal numbers. If you use *xcoloredit* (or a similar program), you must convert the numbers to decimal before adding the color to the *rgb.txt* file. An easy way to perform this conversion is with the UNIX *bc*(1) program, as in the following example.

Say we have a bright shade of blue with the hex value #09E5FB.

- Enter *bc* and press Return.

```
% bc
```

- You can now enter the commands to do the conversion. Set the base of input to 16 (hexadecimal notation has base 16; *bc*'s output defaults to base 10, i.e., decimal).

```
ibase=16
```

- Then enter the hex numbers you want to convert, separated by semicolons; the letters in the hex notations must be in uppercase.

```
09;E5;FB
```

  When you press Return, *bc* gives the decimal values for the three color components (RGB):

```
9
229
251
```

- Type Control-D to quit *bc*.

Note that to convert decimal to hexadecimal, you would skip the `ibase` line; instead specify output as base 16 (`obase=16`); enter the decimal numbers (as you did the hex above); and press Return.

Perhaps a more roundabout way to convert decimal to hex involves the *xtici* client. *xtici* accepts RGB input in decimal notation, but outputs RGB values in hex. See Chapter 8 for details.

Editing and Compiling the Database

Once you have the decimal values for red, green, and blue, you pair them with a color name—we'll call our color tropical blue—and add the color definition to the *rgb.txt* source file. This file is located in the directory *mit/rgb* in the X11 source tree. (Note that you must have write permission for the source files. If you're working on a multi-user system, you'll probably need to speak to the system administrator. If you don't have the source files, consult Volume Eight, *X Window System Administrator's Guide*, for instructions on obtaining them.)

The format of a line in the *rgb.txt* file is:

```
red green blue   color_name
```

where `red`, `green`, and `blue` are integers in the range 0 to 255; the `color_name` is case insensitive but must not include any special characters or symbols. There must be a tab separating the values from the name.

If the color name is composed of two or more words, the color should have two entries, one as multiple words and one as a single word. For example:

```
124 252   0       lawn green
124 252   0       LawnGreen
```

These entries allow you to use either a one- or two-word color name on the command line or in a resource file. (When you use multiple words, they must be surrounded by double quotes. See Chapter 10, *Setting Resources*, for more information about color specifications.)

To update the RGB database, use the following steps:

1. Edit the *rgb.txt* source file (from *mit/rgb*) to add the new color specification(s) (or change existing ones). The new line(s) can go anywhere in the file. Our new color requires the entries:

   ```
   9 229 251       tropical blue
   9 229 251       TropicalBlue
   ```

   If we used a one-word name, a single line would do:

   ```
   9 229 251       caribbean
   ```

2. Run the *rgb* program using the makefile also located in the *mit/rgb* directory. This program converts the text file (*rgb.txt*) to the UNIX *dbm*(1) format files (*rgb.dir* and *rgb.pag*), which are the files actually used as the color database. Just type:

   ```
   % make
   rm -f rgb.pag rgb.dir
   ./rgb rgb < rgb.txt
   ```

3. Then install the new files in */usr/lib/X11* by typing:

   ```
   % make install
   install -c -m 0644 rgb.txt /usr/lib/X11
   install -c -m 0644 rgb.dir /usr/lib/X11
   install -c -m 0644 rgb.pag /usr/lib/X11
   install -c -s showrgb /usr/lib/X11
   install in ./rgb done
   ```

Any colors you've added (or edited) should now be available.

Three alternative color databases are available in the X11 source in the subdirectory *mit/rgb/others*. See "Alternative Release 5 RGB Color Databases" earlier in this chapter and the README file in the source directory for details.

Fixing a Corrupted RGB Database

If the color name database gets corrupted in some way (e.g., written to accidentally), the server may not be able to find any colors with which to display. On a monochrome display, you may get error messages similar to the following:

```
X Toolkit Warning:  Cannot allocate colormap entry for White
X Toolkit Warning:  Cannot allocate colormap entry for Black
```

```
X Toolkit Warning:  Cannot allocate colormap entry for white
X Toolkit Warning:  Cannot allocate colormap entry for black
```

If you get errors of this sort, perform steps 2 and 3 in the procedure described above. This will overwrite the corrupted *rgb* database files.

Creating an Xcms Color Database

Once you have some valid color spaces, creating a Xcms color database is easy—much simpler than editing the RGB database. See Chapter 8, *Other Clients*, for instructions on using the *xtici* color editor, and "The Xcms Color Spaces" earlier in this chapter.

The format of the Xcms database file is:

```
XCMS_COLORDB_START 0.1
color_name<tab>color_space
                .
                .
                .
XCMS_COLORDB_END
```

The first and last lines are literal. Don't forget the `0.1` at the end of the first line; it's important. Between the first and last lines you can put any number of color definitions. The text name goes in the first column, followed by a tab (this is also important), and then a valid color space. Note that the arrangement of the columns is the opposite of that in the *rgb.txt* file, in which the numeric values come first and the text name second.

A system administrator might choose to create an Xcms database that all users can access in the directory */usr/lib/X11* (where *rgb.txt* also generally lives). In this case, the database file should be named *Xcms.txt*. (As we'll see, a user's own Xcms database can have any name.)

Here is a sample database file:

```
XCMS_COLORDB_START 0.1
orchid                  TekHVC:315.8/83.8/24.5
cobalt                  TekHVC:238.3/68.3/30.6
tropical blue           CIEuvY:0.140/0.412/0.615
beach glass             TekHVC:158.2/86.0/15.6
mustard                 CIEuvY:0.227/0.538/0.608
XCMS_COLORDB_END
```

The color names are case-insensitive (for example, "orchid," "Orchid," "ORCHID," and "ORchId" are equivalent).

In our sample database, we've used all portable color specifications, but you can include RGB color spaces as well. (Of course, the RGB specifications will be subject to hardware differences.)

Once *Xcms.txt* is set up, you can specify any of the color names it contains. (Unlike the *rgb.txt* file, no compilation is necessary.) Thus, you can immediately enter the command line:

```
% xsetroot -solid beachglass &
```

Multiple word color names must be specified as a single word or be surrounded by quotes.

By default, Xcms looks for */usr/lib/X11/Xcms.txt*. You can specify an alternative database file by setting the XCMSDB environment variable. This enables every user to have a private color database. Note, however, that Xcms will check only *one* database. If you set XCMSDB to another file, Xcms will not check */usr/lib/X11/Xcms.txt*. (In other words, you cannot specify your own private colors and also take advantage of system-wide Xcms definitions.)

You can call your private Xcms database any name you like; then set the XCMSDB environment variable to the full pathname of the file. For example, say you create some colors you like to use in your normal X session (for the root window, window frames, etc.). You might put these into an Xcms file called *.xcolors*. Then specify:

```
% setenv XCMSDB ~/.xcolors
```

If you're going to be logging on to the same machine consistently, you can put this line in your session file (generally *.xsession* or *.xinitrc*). Of course, you can set XCMSDB to another file any time you like.

Device-Specific Tuning

The X Color Management System provides for display-specific fine tuning in the form of a *Device Color Characterization* (*DCC*) file (also known as a *Device Profile*). The data provided in this file should be specific to the manufacturer, model, size, and screen type of your color monitor. For most users, this fine tuning will be unnecessary. You would probably need two adjacent monitors to perceive the fine adjustments DCC can provide. However, an application developer might choose to install a DCC file.

The DCC data is stored in properties on the screen's root window. Some servers are able to automatically load the properties with data appropriate to the attached display(s). For servers that are built from MIT source, you will probably have to load the DCC data by hand. The *xcmsdb* client that comes with the MIT source distribution will load the DCC data from a text file you specify.

There are two sample DCC files in the directory *mit/clients/xcmsdb/datafiles* , for two types of Tektronix monitors. If you have the MIT X11R5 user-contributed source code available, the directory *contrib/clients/xcrtca/monitors* contains additional DCC files for many commercially available displays:

```
Apollo19.dcc   NWP-513.dcc    Sparc2-19.dcc   VR290.dcc    VR299.dcc
Apple13.dcc    SGI-PI19.dcc   Sun3-60.dcc     VR297-0.dcc
HP98782A.dcc   Sparc1-19.dcc  Trini19.dcc     VR297-1.dcc
```

In addition to these DCC files, the directory contains files with *.ca100* extensions. These files represent an intermediary step between raw color data and the actual DCC files. See Volume Eight, *X Window System Administrator's Guide*, for more information.

The top portion of a DCC file (following some comments) gives a description of the monitor. The following lines appear in the file *Sparc1-19.dcc*:

```
SCREENDATA_BEGIN        0.3

    NAME                Sun SPARCstation 1 19" color monitor
    PART_NUMBER         3
    MODEL               Hitachi HM-4119-S-AA-0, July 1989
```

```
SCREEN_CLASS        VIDEO_RGB
REVISION            2.0
                    .
                    .
                    .
```

The remainder of the file provides data about the monitor's color capabilities. This data is loaded into the root window properties and then plugged into Xcms functions, allowing each device-independent color value to be converted into a device-specific value. You load a DCC file using the program *xcmsdb*. For example, if you have a Hitachi 19" color monitor on your Sun SparcStation 1, you would use the command:

```
% xcmsdb Sparc1-19.dcc
```

You would typically want to load the DCC file in your startup script (commonly *.xinitrc* or *.xsession*). See Appendix A, *System Management*, for guidelines on writing a startup script.

Keep in mind that this type of color correction may be completely unnecessary for you. For more information, see Volume Eight, *X Window System Administrator's Guide*.

13

Customizing mwm

This chapter describes the syntax of the .mwmrc *startup file that can be used to customize the operation of the* mwm *window manager. It describes how to bind functions to keys and how to define your own* mwm *menus. This chapter also explains how to set up* mwm *to use an icon box, a window in which icons on the display can be organized.*

In This Chapter:

13
Customizing mwm

The Motif window manager is one of the more flexible window managers available in the X market today. As we saw in Chapter 3 and Chapter 4, *mwm* provides a wide variety of methods for managing windows (i.e., moving, resizing, iconifying, etc.). In addition, virtually every feature of *mwm* can be customized. You can change the appearance of window frames, icons, and menus, the functions available on the Root Menu and the Window Menu, the keyboard focus policy, how icons are arranged on the display, as well as the appearance of client applications running with *mwm*. As we'll see, you can also create additional menus, displayed from the root window, to perform actions on the display as a whole.

Customization of *mwm* is controlled in two ways:

- Through a special file, called *.mwmrc*, in your home directory.
- Through *mwm* resources you can enter in your *.Xresources* file (or other sources of resource specification).

The default operation of *mwm* is largely controlled by a system-wide file, called *system.mwmrc*, which establishes the contents of the Root Menu and Window Menu, how menu functions are invoked, and what key and button combinations can be used to manage windows. To modify the behavior of *mwm*, you can edit a copy of this file in your home directory. The version of this file in your home directory should be called *.mwmrc*. We'll take a look at the *system.mwmrc* and ways to edit your own *.mwmrc* file to make the window manager work more effectively for you.

In addition to the flexibility provided by the *.mwmrc* file, *mwm* provides dozens of application resources that you can set! It's neither practical nor necessary to discuss all of those resources here. (You could spend quite a long time customizing *mwm*, if you had the time and inclination.) We'll just consider some basic categories into which *mwm* resources can be divided and also look at some of the more useful resources. See Chapter 11 for syntax rules and information about loading resources into the server so that they will be accessible to client programs. See the *mwm* reference page in Part Three of this guide for descriptions of all available resources.

In the remainder of this chapter, we're going to demonstrate the *basics* of customizing *mwm* and suggest what we think are helpful modifications. (This is still quite a lot to absorb.) To illustrate, we'll discuss how to customize the following features of *mwm*:

- The menus and how menu functions are invoked.
- The keyboard focus policy.
- How icons are organized (namely, how to set up a window known as an *icon box*, in which icons on the display can be organized).

Before we can customize the *mwm* menus or the ways in which their functions are invoked, we need to take a closer look at the *system.mwmrc* file. First, however, let's consider an important topic: how to make the window manager aware of customizations.

Activating Changes to the Window Manager

Be aware that if you edit your *.mwmrc* or *.Xresources* file to change the way *mwm* works, the changes will not take effect automatically. Whether you change resource settings, edit your *.mwmrc* file, or both, you must restart *mwm* for the changes to take effect.

If you edit your resources file, you must first make the server aware of the new resource specifications by using the *xrdb* client. Generally, you will enter the following command at the prompt in an *xterm* window:

```
% xrdb -load .Xresources
```

The settings in the current version of your *.Xresources* file will replace the resource settings previously stored in the resource database. You can merely append new settings to the old ones using the *xrdb* `-merge` option. See Chapter 11, *Setting Resources*, for more information.

Once you've loaded the new resource settings, you can restart *mwm*. This can be done using the Restart item of the Root Menu, as described in Chapter 4. When *mwm* has been restarted, it should reflect any changes made to the *.mwmrc* and *.Xresources* files.

Switching Between Custom Version and System Defaults

If you customize any feature of *mwm* and decide you don't like the result, you can restart the window manager with the default settings for your system by typing the keystroke combination:

```
Shift-Control-Meta-!
```

This somewhat involved combination invokes a predefined *mwm* function called `f.set_behavior`, which is a toggle, or switch, between your customized version of *mwm* and the standard system version. If you've customized *mwm*, `f.set_behavior` restarts the window manager using the system defaults. If you then invoke `f.set_behavior` again, the window manager is restarted using your previous customizations.

In either case, a dialog box will ask you to OK or Cancel the process. Click on the appropriate choice with the first pointer button or press Return to select the highlighted button (OK). Figure 13-1 shows the two possible dialog boxes.

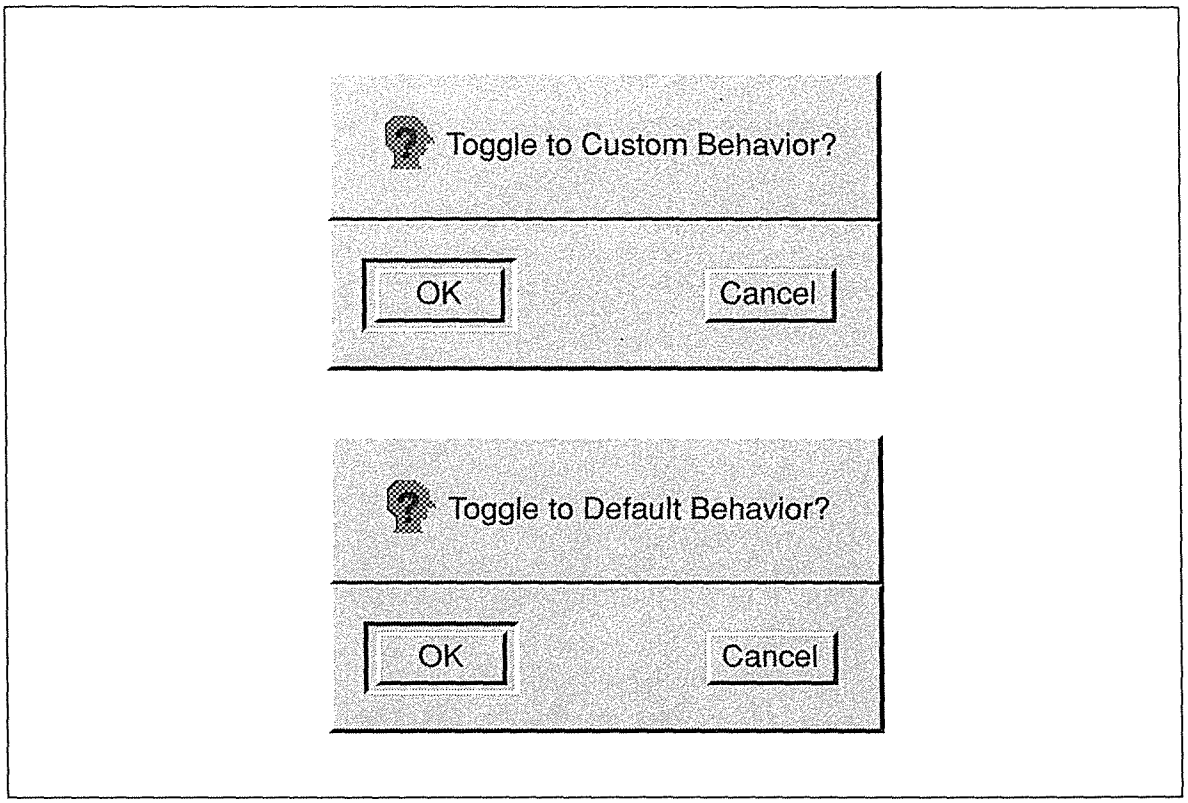

Figure 13-1. Dialog boxes to toggle custom and default mwm environments

The section "mwm Functions" later in this chapter gives an overview of how predefined functions such as `f.set_behavior` work. All of the available functions are described on the *mwm* reference page in Part Three.

The default Root Menu definition (in the standard *.mwmrc* file) includes an item called Toggle Behavior, which invokes `f.set_behavior`. This item is commented out in the standard specification. If you want this item to be available on the Root Menu, it's a simple matter of removing the comment mark from your *.mwmrc* file and restarting the window manager (as described in the preceding section). See the sections "Menu Specifications" and "Customizing the Root Menu" for more information.

A final note: To be truly useful, the Toggle Behavior menu item must be added both to your own *.mwmrc* file *and* to the *system.mwmrc* file. If it only appears in your own *.mwmrc* file, when you select it your environment will be changed to reflect the standard Root Menu—which doesn't offer Toggle Behavior! Then the only way to toggle back is to use the Shift-Control-Meta-! keystroke combination.

The system.mwmrc File

Example 13-1 shows the *system.mwmrc* file shipped with OSF/Motif Release 1.2. If you've used other window managers, this file may seem a bit more complicated than other configuration files, but the complexity is deceptive. A line beginning with an exclamation mark (!) or a number sign (#) is treated as a comment. If a line ends with a backslash (\), the subsequent line is considered a continuation of that line.

If you wish to change the operation of *mwm*, you shouldn't change the *system.mwmrc* file. Instead, copy it to your home directory, under the name *.mwmrc*, and make changes to that copy.

Example 13-1. The system.mwmrc file, Release 1.2

```
!
! (c) Copyright 1989, 1990, 1991, 1992 OPEN SOFTWARE FOUNDATION, INC.
! ALL RIGHTS RESERVED
!
!
! Motif Release 1.2
!

!!
!!  DEFAULT Mwm 1.2 RESOURCE DESCRIPTION FILE (system.mwmrc)
!!
!!  NOTE: To personalize this file, copy this file before editing it.
!!        Personalize copies of the Mwm resource file typically
!!        reside as:
!!
!!                        $HOME/.mwmrc
!!

!!
!! Root Menu Description (this menu must be explicitly posted via f.menu)
!!

Menu DefaultRootMenu
{
        "Root Menu"                     f.title
        "New Window"                    f.exec "xterm &"
        "Shuffle Up"                    f.circle_up
        "Shuffle Down"                  f.circle_down
        "Refresh"                       f.refresh
        "Pack Icons"                    f.pack_icons
!       "Toggle Behavior..."            f.set_behavior
         no-label                       f.separator
        "Restart..."                    f.restart
!       "Quit..."                       f.quit_mwm

}

Menu RootMenu_1.1
{
        "Root Menu"                     f.title
        "New Window"                    f.exec "xterm &"
        "Shuffle Up"                    f.circle_up
        "Shuffle Down"                  f.circle_down
        "Refresh"                       f.refresh
```

Example 13-1. The system.mwmrc file, Release 1.2 (continued)

```
!       "Pack Icons"                    f.pack_icons
!       "Toggle Behavior"               f.set_behavior
        no-label                        f.separator
        "Restart..."                    f.restart
}

!!
!! Default Window Menu Description
!!

Menu DefaultWindowMenu
{
        Restore         _R              Alt<Key>F5          f.restore
        Move            _M              Alt<Key>F7          f.move
        Size            _S              Alt<Key>F8          f.resize
        Minimize        _n              Alt<Key>F9          f.minimize
        Maximize        _x              Alt<Key>F10         f.maximize
        Lower           _L              Alt<Key>F3          f.lower
        no-label                                            f.separator
        Close           _C              Alt<Key>F4          f.kill
}


!!
!! Key Binding Description
!!

Keys DefaultKeyBindings
{
        Shift<Key>Escape                window|icon         f.post_wmenu
        Alt<Key>space                   window|icon         f.post_wmenu
        Alt<Key>Tab                     root|icon|window    f.next_key
        Alt Shift<Key>Tab               root|icon|window    f.prev_key
        Alt<Key>Escape                  root|icon|window    f.circle_down
        Alt Shift<Key>Escape            root|icon|window    f.circle_up
        Alt Shift Ctrl<Key>exclam       root|icon|window    f.set_behavior
        Alt<Key>F6                      window              f.next_key transient
        Alt Shift<Key>F6                window              f.prev_key transient
        Shift<Key>F10                   icon                f.post_wmenu
!       Alt Shift<Key>Delete            root|icon|window    f.restart
}

!!
!! Button Binding Description(s)
!!

Buttons DefaultButtonBindings
{
        <Btn1Down>              icon|frame      f.raise
        <Btn3Down>              icon|frame      f.post_wmenu
        <Btn3Down>              root            f.menu      DefaultRootMenu
}

Buttons ExplicitButtonBindings
{
        <Btn1Down>              frame|icon      f.raise
        <Btn3Down>              frame|icon      f.post_wmenu
        <Btn3Down>              root            f.menu      DefaultRootMenu
!       <Btn1Up>                icon            f.restore
```

Example 13-1. The system.mwmrc file, Release 1.2 (continued)

```
        Alt<Btn1Down>           window|icon         f.lower
!       Alt<Btn2Down>           window|icon         f.resize
!       Alt<Btn3Down>           window|icon         f.move

}

Buttons PointerButtonBindings
{
        <Btn1Down>              frame|icon          f.raise
        <Btn3Down>              frame|icon          f.post_wmenu
        <Btn3Down>              root                f.menu      DefaultRootMenu
        <Btn1Down>              window              f.raise
!       <Btn1Up>                icon                f.restore
        Alt<Btn1Down>           window|icon         f.lower
!       Alt<Btn2Down>           window|icon         f.resize
!       Alt<Btn3Down>           window|icon         f.move
}

!!
!!  END OF mwm RESOURCE DESCRIPTION FILE
!!
```

The *system.mwmrc* file can be divided into three sections:

- Menu specifications.
- Key bindings.
- Button bindings.

The menu section of the *system.mwmrc* file defines the contents of the Root Menu and the Window Menu. Menu item labels are paired with predefined *mwm* functions.

A *binding* is a mapping between a user action (such as a keystroke) and a function, in this case a window manager function. The key bindings section specifies keyboard keys that can be used to invoke some of the predefined window manager functions. The button bindings section specifies pointer buttons or key/button combinations that can be used to invoke various functions.

Each section of the *system.mwmrc* file matches the following basic template:

```
Section_Type   Section_Title
{
definitions
}
```

For example, the basic syntax of a menu specification is as follows:

```
Menu menu_name . . .
{
menu items defined
}
```

Menu is the *Section_Type*. The other possible section types are Keys and Buttons. The *Section_Title* is somewhat arbitrary. In this case, it corresponds to the title of a menu. In the key and button sections, it is simply a title assigned to a group of bindings.

However, the *Section_Title* can be very significant. As we'll see, a section title can be used as the value of a resource variable in your *.Xresources* file. Menu titles are often referenced elsewhere in the *.mwmrc* file. The *menu_name* is generally paired with a pointer button action (in the button bindings section of the *.mwmrc* file) to allow you to use a particular button to display the menu.

The syntax of the actual menu items, key bindings, and button bindings requires further explanation. But first, let's take a look at some of the predefined window manager functions.

mwm Functions

mwm has a number of predefined functions. Each of these functions has a name beginning with "f.". Several functions appear in the *system.mwmrc* file, paired with the method by which the function can be invoked: by menu item, pointer button action, keystroke(s), or key and pointer button combinations.

The meaning of most of these functions should be fairly obvious to you from the name, if not from your experience using the window manager. For example, f.resize is used to resize a window, f.move to move a window, and f.minimize to change a window to an icon.

Others are less obvious. The function f.post_wmenu is used to display (or post) the Window Menu. Notice the function f.separator, which appears in the menu definition coupled with the instruction no-label rather than with a menu item. This line in the *.mwmrc* creates a divider line on a menu. For example, such a divider line is used to isolate the Restart item from the other items on the Root Menu.

As we'll see, the function f.menu is used to associate a menu with the key or button binding that is used to display it. The f.menu function takes a required argument: the menu name. This function can also be used to define a submenu.

Each of the functions is described in detail on the reference page for *mwm* in Part Three of this guide.

Menu Specifications

The first section of the *system.mwmrc* file contains specifications for the Root Menu and Window Menu. As we've said, the basic syntax of a menu specification is as follows:

```
Menu menu_name . . .
{
menu items defined
}
```

As you may notice, the *system.mwmrc* file defines two different "Root Menus." The first version has the *menu_name* DefaultRootMenu and the second has the name Root-Menu_1.1. The DefaultRootMenu is just that—the Root Menu you get by default and

the one described in Chapter 4, *More about the mwm Window Manager*. As we'll see, the *menu_name* is paired with either a pointer button action or a keystroke combination—in the button or key bindings section of the *.mwmrc* file—to allow you to use the button or key to display the menu. As described in Chapter 4, the `DefaultRootMenu` is displayed by placing the pointer on the root window and holding down the third pointer button.

The `RootMenu_1.1` reiterates the Root Menu definition provided with *mwm* 1.1. The *system.mwmrc* file does not bind this menu to a key or button, so there is no way to display it without customization. Hypothetically, you could edit your *.mwmrc* to be able to display the `RootMenu_1.1` using a different key than the one bound to the `DefaultRootMenu`—and thus, have access to *both* versions. However, since the default version provides all of the functionality of the earlier one, there isn't much point. For the purposes of our discussion, we'll just consider the default Root Menu.

Menu items are defined in slightly different ways for the Root Menu and the Window Menu. The following text in the *system.mwmrc* file creates the default Root Menu:

```
# Root Menu Description
Menu DefaultRootMenu
{
        "Root Menu"                     f.title
        "New Window"                    f.exec "xterm &"
        "Shuffle Up"                    f.circle_up
        "Shuffle Down"                  f.circle_down
        "Refresh"                       f.refresh
        "Pack Icons"                    f.pack_icons
!       "Toggle Behavior..."            f.set_behavior
         no-label                       f.separator
        "Restart..."                    f.restart
!       "Quit..."                       f.quit_mwm
}
```

The syntax for defining Root Menu items is very simple. Each item is defined by a line of this format:

"label" function

When you pair a label with a menu function, that label appears as a menu item. You can invoke the function by selecting the item from the menu using the pointer. For example, the line:

```
"Refresh"       f.refresh
```

sets up the Refresh menu item, which can be selected from the Root Menu as discussed in Chapter 4. (Again, the function performed is obvious from the function name.) As we'll see later, it's easy to add items to the Root Menu by adding lines of label/function pairs. (You can also use a bitmap image rather than a text label, if you like.)

Notice that two Root Menu items are commented out.

```
!     "Toggle Behavior..." f.set_behavior
!     "Quit..."       f.quit_mwm
```

These items will not appear on your Root Menu. We'll discuss adding these items in the section "Customizing the Root Menu" later in this chapter.

Because Window Menu items can be invoked in a variety of ways, the syntax for defining items is more complicated. The following text defines the Window Menu:

```
# Default Window Menu Description

Menu DefaultWindowMenu
{
    Restore          _R          Alt<Key>F5          f.restore
    Move             _M          Alt<Key>F7          f.move
    Size             _S          Alt<Key>F8          f.resize
    Minimize         _n          Alt<Key>F9          f.minimize
    Maximize         _x          Alt<Key>F10         f.maximize
    Lower            _L          Alt<Key>F3          f.lower
    no-label                                         f.separator
    Close            _C          Alt<Key>F4          f.kill
}
```

The syntax of each menu item is as follows:

"label" *mnemonic* *accelerator* *function*

(The *mnemonic* and *accelerator* fields are optional.) Like the Root Menu, each item on the Window Menu can be invoked by selecting its label with the pointer. In addition, there are two shortcuts defined for invoking the function: a mnemonic and an accelerator. As you may recall, a mnemonic is a unique letter abbreviation for the menu item label. On the menu, mnemonic abbreviations are underlined; thus an underscore precedes each mnemonic definition in the *system.mwmrc* file. Once the Window Menu is displayed, you can select an item by typing its mnemonic abbreviation. Similarly, you can invoke the function without displaying the menu, simply by typing the accelerator keys (by default, the Alt key plus a function key).*

Now let's see how one of the Window Menu definition lines fits this template:

```
Move    _M              Alt<Key>F7      f.move
```

The menu item label is Move. Selecting the item invokes the `f.move` function. The mnemonic "m" or the accelerator key combination Alt-F7 can also be used to invoke the function.

Key Bindings

The second section of the *system.mwmrc* file binds keystroke combinations to window manager functions.

Like the menu definition section, the key bindings section of the file is titled and bracketed:

```
Keys Section_Title
{
key bindings defined
}
```

*If your keyboard does not have an F10 function key, you cannot use the accelerator for the Maximize item without doing some customization. A possible workaround is to edit the line defining the Maximize menu item in your *.mwmrc* file. Changing F10 to F2 will suffice in most cases.

The section type is `Keys`. The section title in the *system.mwmrc* file is `DefaultKeyBindings`. This title can also be specified as the value of the *mwm* resource `keyBindings` in your *.Xresources* file. However, since these bindings are used by default, this is not necessary.

Using the section title as a resource becomes significant when you want to create an alternative set of bindings. Hypothetically, you could add another set of bindings with a different title to your *.mwmrc* file. Then specify this title as the value of the `keyBindings` resource in your *.Xresources* file. If you add the following resource specification to your *.Xresources* file, `MyButtonBindings` replace `DefaultButtonBindings` for all client applications running with *mwm*:

```
Mwm*keyBindings:        MyButtonBindings
```

If you want to use different sets of bindings for different applications, you can add an application name between the parts of the resource specification. For example, if you want `MyButtonBindings` to apply only to *xterm* windows running with *mwm*, you could enter the following resource line:

```
Mwm*xterm*keyBindings:          MyButtonBindings
```

Then `DefaultButtonBindings` would still apply to all applications other than *xterm*.

A non-obvious principle behind a key/function (or button/function) binding is that in order for the keys (or buttons) to invoke the function, the pointer must be in a certain location. This location is known as the *context*. For key bindings, the useful contexts are: `root`, `window`, and `icon`. The `window` context refers to the entire window, including the frame. (There are other more specific contexts, such as `border`, explained under "Button Bindings," but when specifying key bindings, these contexts are all equivalent to `window`.)

Some functions can be invoked if the pointer is in more than one context. For example, as we saw in Chapter 4, you can display the Window Menu from either a window or an icon using the keyboard shortcuts Meta-space or Shift-Escape. The action involved is `f.post_wmenu` and the window and the icon are the pointer contexts from which this action can be performed. These keyboard shortcuts are defined in the key bindings section of the *system.mwmrc* file as follows:

```
Shift<Key>Escape      window|icon      f.post_wmenu
Meta<Key>space  window|icon      f.post_wmenu
```

Upon examining these lines, we can discern the template for a key binding:

[*modifier_keys*]<Key>*key_name* *context* *function*

Each binding can have one or more modifier keys (modifiers are optional) and *must* have a single primary key (signaled by the word <Key> in angle brackets) to invoke the function. In the first specification, Shift is the modifier and Escape is the primary key. In the second specification, Meta is the modifier and space is the primary key. Both specifications have two acceptable pointer contexts: either a window or an icon. And both bindings are mapped to the same action, `f.post_wmenu`, which displays the Window Menu.

Button Bindings

The key bindings section of the file is also titled and bracketed:

```
Buttons Section_Title
{
button bindings defined
}
```

The section type is `Buttons`. The *system.mwmrc* file contains three sets of button bindings with the section titles:

```
DefaultButtonBindings
ExplicitButtonBindings
PointerButtonBindings
```

Button bindings clearly illustrate the need to coordinate your *.Xresources* and *.mwmrc* files. The three sets of button bindings correspond to three possible settings for the resource `buttonBindings`. The default setting for the resource is:

```
Mwm*buttonBindings:  DefaultButtonBindings
```

specifying that the `DefaultButtonBindings` are used.

You can specify that one of the other sets of button bindings is to be used by setting this resource in your *.Xresources* file. For example, if you add the following specification to your resource file:

```
Mwm*buttonBindings:  ExplicitButtonBindings
```

mwm will use those bindings that come under the heading `ExplicitButtonBindings` in the *.mwmrc* file.

Be aware that if you do specify different button bindings, the value of the resource must exactly match the title associated with the bindings, or the bindings will not take effect.

The syntax for a button binding specification is very similar to that of a key binding:

[modifier_key]<button_event> context function

Each button binding can have one or more modifier keys (modifiers are optional) and *must* have a single button event (enclosed in angle brackets) to invoke the function. The motion that comprises each button event should be fairly obvious. (Lists of acceptable button events and modifier keys appear on the *mwm* reference page in Part Three of this guide.)

For button bindings. the valid contexts are `root`, `window`, `icon`, `title`, `border`, `frame`, and `app`. The `title` context refers to the title area of the frame. `border` refers to the frame exclusive of the titlebar. `frame` refers to the entire frame (thus it encompasses `title` and `border`). The `app` context refers to the application window proper (i.e., exclusive of the frame). The `window` context includes the application window and the frame (thus it encompasses `app`, `frame`, `border`, and `title`).

Now let's see how the button binding syntax relates to the default button bindings in the *system.mwmrc* file:

```
Buttons DefaultButtonBindings
{
    <Btn1Down>          icon|frame      f.raise
    <Btn3Down>          icon            f.post_wmenu
    <Btn1Down>          root            f.menu      RootMenu
}
```

The first specification is familiar. It indicates that the event of pressing down the first pointer button while the pointer is in a window frame or an icon performs the action of raising the window or icon, respectively.

The second binding reveals *still another* way to display the Window Menu, by pressing the third pointer button on an icon.

The third binding is also familiar and illustrates the use of the `f.menu` function. As previously mentioned, the `f.menu` function is used to associate a menu with the key or button binding that is used to display it. The following binding specifies that the Root Menu is displayed by pressing and holding down the first pointer button on the root window:

```
<Btn1Down>          root            f.menu      RootMenu
```

Notice that the function requires an argument, the menu name (RootMenu), which also appears in the first line of the menu definition. This correspondence is required—`f.menu` needs to know which menu to display.

Customizing the Root Menu

You can add items to the Root Menu simply by adding lines of the format:

"label" *function*

within the menu definition section of your *.mwmrc* file (and then restarting the window manager).

The `f.exec` function allows you to execute system commands from a menu. In the default Root Menu, the New Window command uses the `f.exec` function to execute the system command `xterm &`, as shown below:

```
# Root Menu Description
Menu DefaultRootMenu
{
        "Root Menu"                 f.title
        "New Window"                f.exec "xterm &"
        "Shuffle Up"                f.circle_up
        "Shuffle Down"              f.circle_down
        "Refresh"                   f.refresh
        "Pack Icons"                f.pack_icons
!       "Toggle Behavior..."        f.set_behavior
         no-label                   f.separator
        "Restart..."                f.restart
!       "Quit..."                   f.quit_mwm
}
```

To create a menu item labeled Clock that opens an *xclock* window on your display, simply add a line to your *.mwmrc* file, as shown here:

```
# Root Menu Description
Menu DefaultRootMenu
{
        "Root Menu"                 f.title
        "New Window"                f.exec "xterm &"
        "Clock"                     f.exec "xclock &"
        "Shuffle Up"                f.circle_up
        "Shuffle Down"              f.circle_down
        "Refresh"                   f.refresh
        "Pack Icons"                f.pack_icons
!       "Toggle Behavior..."        f.set_behavior
         no-label                   f.separator
        "Restart..."                f.restart
!       "Quit..."                   f.quit_mwm
}
```

In most cases, the label is a text string. However, you can use a bitmapped image instead by preceding it with an "at" symbol (@). The following line lets you run *xbiff* by selecting the bitmap image of a full mailbox (filename *flagup*) from the menu.

```
@flagup         f.exec "xbiff &"
```

Unless a full pathname is given for the bitmap file, *mwm* looks for bitmap files in a system-wide directory, generally */usr/include/X11/bitmaps*. (*flagup* is a standard bitmap available in */usr/include/X11/bitmaps*. It's the image *xbiff* uses for its full mailbox window.) You can also specify an alternate standard bitmap directory using the `bitmapDirectory` resource. If a bitmap is not found in the standard bitmap directory and the XBMLANGPATH environment variable is set, *mwm* checks that directory.

You can also edit (or remove) existing menu items. As we pointed out earlier, two items (Toggle Behavior and Quit) are commented out. Toggle Behavior invokes `f.set_behavior`, which toggles between your own customized version of *mwm* and the standard version for your environment. (You can also invoke this function using the key combination Shift-Control-Meta-!. See the section "Switching between Custom Version and System Defaults" earlier in this chapter for more information.) Quit causes the window manager to exit (it is not restarted). You might invoke Quit before starting another window manager, such as *twm*.

To add either Toggle Behavior or Quit to the Root Menu, just delete the initial exclamation mark, which comments out the line (and restart the window manager, as usual).

You might also want to change what an existing menu item does. Say you want to run the *hpterm* terminal emulator (developed by Hewlett-Packard) rather than *xterm*. You would edit the New Window line in your menu specification to look like this:

```
# Root Menu Description
Menu DefaultRootMenu
{
        "Root Menu"                 f.title
        "New Window"                f.exec "hpterm &"
        "Clock"                     f.exec "xclock &"
        "Shuffle Up"                f.circle_up
        "Shuffle Down"              f.circle_down
        "Refresh"                   f.refresh
```

```
        "Pack Icons"                f.pack_icons
!       "Toggle Behavior..."        f.set_behavior
        no-label                    f.separator
        "Restart..."                f.restart
!       "Quit..."                   f.quit_mwm
}
```

Creating New Menus

Keep in mind that *mwm* also allows you to specify entirely new menus in your *.mwmrc* file. A new menu can be separate from all existing menus, or it can be a submenu of an existing menu. (Submenus are described in the following section, "Cascading Menus.")

If you want to create a new, independent menu, it must conform to the menu specification syntax discussed earlier. Items must invoke predefined window manager functions.

The *.mwmrc* file must also specify how the menu will be displayed and in what context. This involves associating a key or button with the `f.menu` function. Say you've specified a new menu, titled GamesMenu, that runs various game programs, each in its own window. (The `f.exec` function would be used to define each item.) The following button binding specifies that pressing the second pointer button on the root window displays the Games Menu:

```
<Btn2Down>          root        f.menu      GamesMenu
```

Cascading Menus

mwm also allows you to create submenus, generally known as *cascading* menus because they are displayed to the right side of (and slightly lower than) another menu. You define a submenu just as you would any other, using the syntax rules discussed earlier. The following lines create a Utilities Menu that invokes several "desktop" clients and one game:

```
Menu UtilitiesMenu
{
    "Utilities Menu"      f.title
    "Clock"               f.exec "xclock &"
    "System Load"         f.exec "xload &"
    "Calculator"          f.exec "xcalc &"
    "Manpage Browser"     f.exec "xman &"
    "Tetris"              f.exec "xtetris &"
}
```

In order to make the Utilities Menu a submenu of the Root Menu, you need to add an `f.menu` function to the Root Menu. This `f.menu` function must be coupled with the correct submenu title:

```
# Root Menu Description
Menu DefaultRootMenu
{
        "Root Menu"                 f.title
        "New Window"                f.exec "hpterm &"
        "Shuffle Up"                f.circle_up
        "Shuffle Down"              f.circle_down
        "Refresh"                   f.refresh
```

```
        "Pack Icons"                f.pack_icons
        "Utilities"                 f.menu              UtilitiesMenu
!       "Toggle Behavior..."        f.set_behavior
         no-label                   f.separator
        "Restart..."                f.restart
!       "Quit..."                   f.quit_mwm
}
```

After you specify the preceding menus in your *.mwmrc* file (and restart *mwm*), display the Root Menu. It will feature a new item, labeled Utilities. Since this item is actually a pointer to a submenu, it will be followed by an arrowhead pointing to the right, as in Figure 13-2.

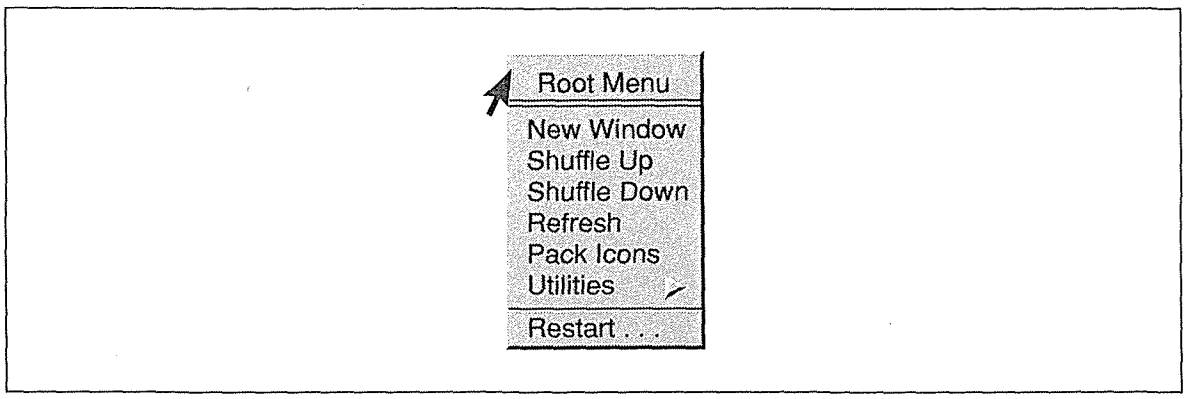

Figure 13-2. An arrowhead pointing to the right indicates a submenu

If you drag the pointer down the Root Menu to the Utilities item, the submenu will appear to cascade to the right. Figure 13-3 shows it appearing.

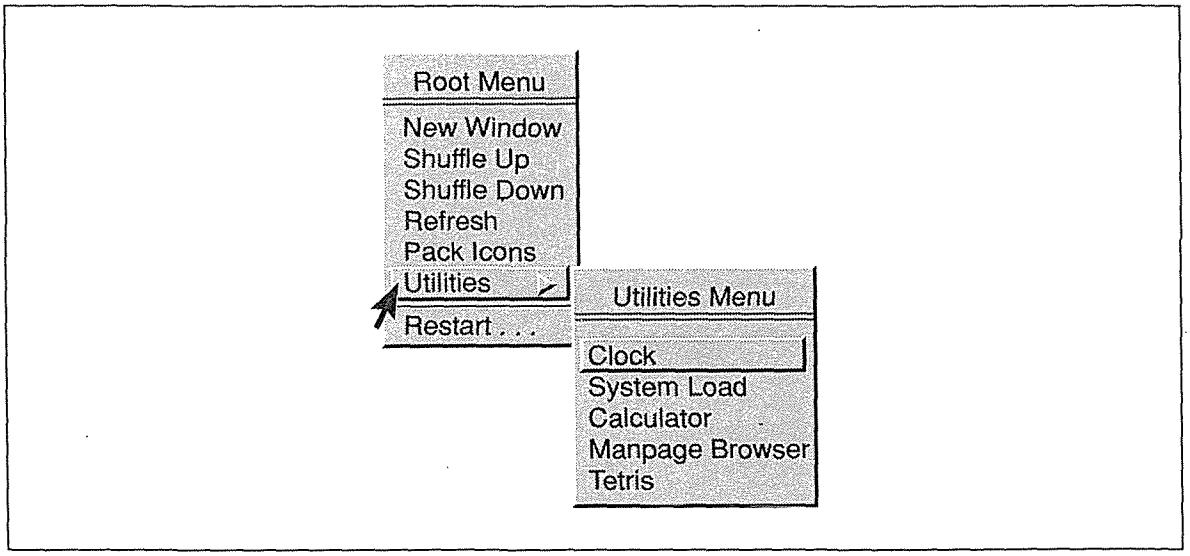

Figure 13-3. Utilities submenu of the Root Menu

If you release the pointer button, both menus will remain displayed and the Utilities item and the first item on the Utilities Menu will be highlighted by a box. You can then select an item from the Utilities Menu by moving the pointer to the item and clicking the first button.

Keep in mind that you can create several submenus beneath a single menu and that menus can cascade several levels, though such complexity is not necessarily desirable.

Note also that if you pair a label with an invalid function (or with `f.nop`, which specifies no operation), or with a function that doesn't work in the current context, the label appears in a lighter typeface. This "graying out" indicates that the menu item is not available for selection.

Setting mwm Resources

The Motif window manager provides dozens of resources that control the appearance and functionality of the window manager, its component features, and other clients running with it. *mwm* resources should be entered in your *.Xresources* file and will take effect when the resources have been loaded into the server and *mwm* has been started or restarted. See Chapter 11, *Setting Resources*, for syntax information and instructions on how to load resources using the *xrdb* client. See "Activating Changes to the Window Manager" earlier in this chapter for information about running *mwm* with the new resource settings.

mwm resources are considered to fall into three categories:

1. *mwm* component appearance resources. These resources set the characteristics of *mwm*'s component features, such as the window frame, menus, and icons.
2. *mwm-specific* appearance and behavior resources. These resources set characteristics of the window manager client, such as focus policy, key and button bindings, and so forth.
3. *Client-specific* resources. These *mwm* resources can be used to set the appearance and behavior of a particular client or class of clients.

Under these categories fall dozens of *mwm* resources. The sheer number of resources makes it impractical for all of them to be discussed here. In the following sections, we discuss the three categories of resources in somewhat greater detail. We'll then take a look at two of the more powerful and useful resources, `keyboardFocusPolicy` and `useIconBox`, which set the focus policy and set up *mwm* to use an icon box, respectively. For a comprehensive list of available resources, see the *mwm* reference page in Part Three of this guide. Note that the reference page groups the resources according to the categories we've listed.

Component Appearance Resources

The Motif window manager can be considered to be made up of components: client window frames, menus, icons, and what are known as *feedback* or *dialog boxes*. An example of a feedback box is the box that appears so that you can confirm or cancel a Restart command from the Root Menu. (See "The Root Menu" section in Chapter 4, *More about the mwm Window Manager*.)

Certain resources allow you to specify the appearance of one or all of these *mwm* component features. In specifying the resource setting, you can use the name of one of the features as part of the resource name. For example, one of the most useful component appearance resources is `background`, which, as we know from Chapter 11, specifies the background color. You can specify a resource that sets the background color of any of the *mwm* components. The following resource specification sets the background color of all client window frames to light blue:

```
Mwm*client*background:      lightblue
```

Table 13-1 summarizes the resource name that corresponds to each of the *mwm* components.

Table 13-1. Resource Names Corresponding to mwm Components

Component	Resource name
Menu	`menu`
Icon	`icon`
Client window frame	`client`
Feedback/dialog box	`feedback`
Titlebar	`title`

Thus, to set the background color of feedback boxes to sea green, you'd use the following resource:

```
Mwm*feedback*background:  seagreen
```

Of course, if you omit any specific component from the resource specification, it applies to *all* components. Thus, the following specification sets the background color of all window frames, feedback boxes, icons, and menus to light grey:

```
Mwm*background: lightgrey
```

Since the titlebar is actually part of the client window frame, the `title` resource is a special case. (Technically speaking, `title` comes at a different level in the widget hierarchy than `client` and the other component resources.) The `title` resource allows you to specify characteristics for the titlebar alone (including the command buttons), while you can specify characteristics for the rest of the frame (the resize border) using `client`. Thus, you might have the resource specifications:

```
Mwm*client*title*background:    lightblue
Mwm*client*background:    aquamarine
```

These lines would create two-tone window frames with aquamarine borders and light blue titlebars (perhaps too vivid a combination). (Note that these colors do not apply to the active (focus) window. To change these characteristics for the focus window, you would need to specify the `activeBackground` resource. See the *mwm* reference page in Part Three of this guide for more information.)

Similarly, you can specify resources for individual menus, by using the `menu` component with the menu name, as in the following example:

```
Mwm*menu*UtilitiesMenu*background:    seagreen
```

This line gives the Utilities Menu we added to our *.mwmrc* file a sea green background. (See "Cascading Menus" earlier in this chapter for information about creating submenus.)

mwm-specific Appearance and Behavior Resources

The *mwm*-specific resources control aspects of what you probably think of as the window manager application itself, features such as the focus policy, whether windows are placed on the display automatically or interactively, which set(s) of button and key bindings are used, whether an icon box is used, and so forth.

The syntax of *mwm*-specific resource specifications is very simple—the *mwm* class name connected by a loose binding to the resource variable name, as shown here:

```
Mwm*clientAutoPlace: false
```

This resource establishes the behavior that the user will interactively place client windows on the display. (The default is true, meaning *mwm* places them automatically.)

Two of the *mwm*-specific resources bring up an issue of coordination between the *.Xresources* and *.mwmrc* files. Remember, the default *.mwmrc* file contains three sets of button bindings:

```
DefaultButtonBindings
ExplicitButtonBindings
PointerButtonBindings
```

These three sets of button bindings correspond to three possible settings for the resource variable `buttonBindings`. If your resource file contains the following setting:

```
Mwm*buttonBindings:  ExplicitButtonBindings
```

mwm will use those bindings that come under the heading `ExplicitButtonBindings` in the *.mwmrc* file.

Similarly, the resource variable `keyBindings` should be coordinated to match the key bindings in the *.mwmrc* file. Since the default *.mwmrc* file has only one set of key bindings, named `DefaultKeyBindings`, and the `keyBindings` resource also sets this by default, coordination should not be an issue unless you create a new set of key bindings with a different name.

Two of the most useful and powerful *mwm*-specific resources set the keyboard focus policy and specify that icons be stored in an icon box. We'll discuss the use and advantages of these resources later in this chapter.

Client-specific Resources

Some *mwm* resources can be set to apply to certain client applications or classes of applications. These resources generally have the form:

```
Mwm*application*resource_variable:
```

where `application` can be an instance name or a class name. Be aware that the application name is optional. If you omit an application name, the resource applies to all clients. (Client-specific resource specifications take precedence.)

In rare cases, an application might not have instance and class resource names that are known to the window manager. To see if the window manager knows these names, run *xprop* and click the pointer on the application window. The property WM_CLASS should contain both names. You can specify resources for clients that have unknown instance and class names by using the literal parameter `defaults` in place of a particular `application`.

```
Mwm*defaults*resource_variable:
```

Many of the client-specific resources provide what might be considered advanced customization. For example, a combination of resources allows you to specify your own bitmap as the image for a client icon. Other resources allow you to suppress certain features of the window frame for particular clients. For instance, you may choose to omit the Maximize button from the frame surrounding *xterm* windows. Still others allow you to specify *extra* window decoration in the form of a matte which lies between the application window and the frame. This matte is purely aesthetic; it provides no functionality. The average user will probably not need most of these client-specific resources.

One client-specific resource users might be interested in is called `focusAutoRaise`. This resource causes a window to be raised to the top of the stack when it is selected as the focus window. When the focus policy is explicit (click-to-type), `focusAutoRaise` is true for all clients by default. When the focus policy is pointer (real-estate-driven), `focusAutoRaise` is false for all clients by default.

These defaults are very sensible. If you are using the default click-to-type focus, `focusAutoRaise` is clearly very desirable. You click on a window to focus input and the window is raised to the top of the stack so that you can work with it easily. However, if you change the focus policy to pointer focus (as we'll describe in the following section), turning `focusAutoRaise` on can make the display seem chaotic.

When pointer focus is active as you move the pointer across the display, the focus changes from window to window based on the location of the pointer, often a desirable feature. However, if `focusAutoRaise` is set to be true, each time the pointer moves into a window, the window will be moved to the front of the display. Simply moving the pointer across a screenful of windows can create a distracting shuffling effect! If you set the focus policy to pointer, we suggest you leave `focusAutoRaise` set to false.

Of course, using pointer focus without `focusAutoRaise` is just our preference. You may want to experiment awhile to see how you like working with it.

Hypothetically, you *can* turn `autoFocusRaise` behavior on or off only for particular clients, but this is not necessarily desirable, with either focus policy. For instance, say you're using the default *mwm* settings so that explicit focus is in effect and `focusAutoRaise` is

true for all clients. You can suppress the auto-raise feature only for the class of *xterm* windows by specifying:

```
Mwm*XTerm*focusAutoRaise: false
```

But what is the point? In most cases, you want to raise the focus window so that you can work with it more easily.*

When pointer focus is in effect, setting `focusAutoRaise` differently for different clients can have tedious and unnecessary complications. It becomes fairly easy to "bury" one window beneath another inadvertently. For example, say `focusAutoRaise` is turned on for *xterm* windows only, and turned off for *xbiff*. If an *xbiff* window appears on top of an *xterm* and you move the pointer into the *xterm*, the *xterm* is raised automatically, covering the *xbiff* window.

You can send the *xterm* to the back using the Lower item of the Window Menu. Although the *xterm* retains the focus, it is not raised. `focusAutoRaise` specifies that a window is raised when the focus is moved to a window (retaining the focus is a different matter). However, if you move the pointer to another window and back to the *xterm*, the *xbiff* window will be buried again. In order to avoid such a situation, you would have to arrange all windows so that a part of the frame is exposed at all times. No window should ever appear entirely on top of another.

Given the limitations and potential problems, **we discourage setting `focusAutoRaise` differently for different applications, regardless of the focus policy**.

See the *mwm* reference page in Part Three for descriptions of all of the client-specific resources.

Setting the Focus Policy

The most common resource users will probably want to set controls *mwm*'s keyboard focus policy. By default, *mwm* has explicit (or click-to-type) focus, which is set using the following resource:

```
Mwm*keyboardFocusPolicy:  explicit
```

To change the keyboard focus policy from explicit to pointer focus (that is, focus follows the movement of the pointer), enter the following line in your *.Xresources* file:

```
Mwm*keyboardFocusPolicy:  pointer
```

*Note that even if you turn off the auto-raise feature for *xterm*, it is still possible to raise an *xterm* and select it to receive input simultaneously, but in a more restricted way. Clicking anywhere on a window selects that window to receive the focus. Clicking on the frame, exclusive of the command buttons, raises a window. Thus, by clicking this part of the frame, you can perform both actions simultaneously. However, why restrict yourself to using only a part of the frame, when you can use the entire window?

Using an Icon Box

One of the most interesting (and desirable) features *mwm* can provide is a window in which icons can be organized on the display. This window is known as an *icon box*, and is pictured in Figure 13-4. As we'll see, in addition to organizing icons neatly on the display, the icon box also provides a few window management functions.

You can set up *mwm* to provide an icon box automatically by specifying the following resource in your *.Xresources* file:

```
Mwm*useIconBox: true
```

If this resource is included in your *.Xresources* file (and the resources have been loaded as described in Chapter 11) *mwm* will provide an icon box when it is started (or restarted). Other resources can be used to customize the size, appearance, and location of the icon box, as well as the window's title. By default, the icon box is six icons wide by one icon high and is located in the lower-left corner of the display.

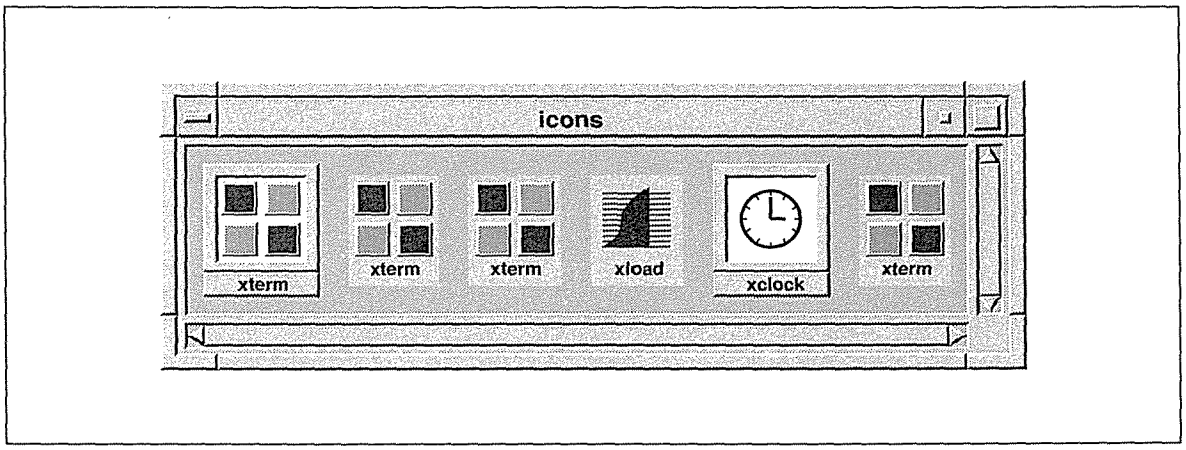

Figure 13-4. An icon box

The horizontal and vertical scrollbars within the icon box suggest a significant, albeit not an obvious, feature. Icons can extend beyond the visible bounds of the icon box. If more than six icons are present in the default size box, you can view them using the scrollbars. (See Chapter 9, *Working with Motif Applications*, for instructions on using a Motif scrollbar.) Keep in mind that if icons do extend beyond the visible bounds of the box, the appearance of the scrollbars will indicate it.

The presence of an icon box changes the way icons are used on the display. If you are using *mwm* without an icon box, only those windows that have been iconified are represented by icons on the display. If you are using *mwm* with an icon box, *all* windows on the display are represented by icons that are stored in the box, whether or not the windows are in an iconified state.

When a client window is started, the window appears on the display *and* a corresponding icon appears in the icon box. However, an icon that represents a window currently visible on the display has a different appearance than an actual icon (that is, an iconified window). An

icon corresponding to a window currently on the display appears flatter and less defined than the image of an iconified window. The former probably has fewer lines in its outer border. If you set up *mwm* to use an icon box, the differing appearance of these two types of icons should be obvious.

Somewhat similar to a menu item in a lighter typeface, the flatter, less defined icon suggests that it is not available to be chosen. In a sense, this is true. Since the flat icon is not an iconified window, but merely an image, it is not available to be converted back to a window. The icon box in Figure 13-4 contains two iconified windows (*xclock* and the first *xterm*) and four icons representing windows currently visible on the display.

You can perform some window management functions by clicking on icons in the icon box. If you double click on an iconified window using the first pointer button, the icon is converted back to a window (and is raised to the top of the stack). If you double click on an icon representing an active window on the display, the corresponding window is raised to the front of the display. (We find this latter function to be not particularly useful.) When you raise a window by clicking on its icon, the icon box retains the focus.

When performing either function, between the first and second clicks you'll probably notice that the Window Menu is displayed for an instant above the icon. If you pause too long between the two clicks in either of these functions, the action you intend (either deiconifying or raising) will fail and the Window Menu will remain on the screen.

If you get stuck on the Window Menu when trying to convert an icon to a window, place the pointer on the Restore menu item and click the first pointer button.

If you get stuck on the Window Menu when trying to raise a window on the display (by clicking on its icon), the menu affords no item to complete the action. Instead you should move the pointer onto the root window and click the first button—the menu will be removed. Then try double clicking again. Or raise the window simply by clicking the first pointer button on the window's frame (exclusive of the frame's command buttons).

As these actions suggest, you can display the Window Menu from any of the icons in the box by clicking the first pointer button on the icon image.* (You can also display the Window Menu from and manage the icon box itself, as we discuss later on.) Depending on whether you click on an iconified window or an icon representing an active window on the display, different Window Menu items are available for selection.

When you display the Window Menu from an iconified window within the icon box, the items Restore, Move, Maximize, and Close are available for selection. Be aware that the Move menu item only allows you to move the icon itself—to another location within the icon box. The other available items perform their standard functions, which are described in the section "Using the Window Menu on Icons" in Chapter 4.

Displaying the Window Menu from an icon representing an active window on the display is not particularly useful: only the items Move and Close are available for selection. And again, the Move menu item only allows you to move the icon within the icon box.

*The 1.0 version of *mwm* was documented to display the Window Menu from an icon within the icon box, but the program did not seem to work according to the specifications. The 1.1 version of *mwm* does provide this functionality.

When you display the Window Menu from the icon box, the menu commands apply to the box itself (which is actually a window). You can display the menu from the icon box using any of the methods described in the section "Using the Window Menu" in Chapter 4. For example, if you use the keyboard shortcut Meta-space, the menu is displayed above the Window Menu command button in the upper-left corner of the icon box frame.

When displayed from the icon box, the Window Menu Close item is replaced by an item called Pack Icons (mnemonic "p", accelerator Shift+Alt+F7). Pack Icons rearranges the icons in the box to fill in empty slots. This is useful when icons are removed from the box or the box is resized.

When you remove a window, the corresponding icon is removed from the box, leaving an empty slot. Pack Icons will move any icons that are to the right of the slot one space to the left to fill the hole. If you resize the icon box, Pack Icons will arrange the icons to fit the new window in an optimal way. For instance, say we resize the icon box in Figure 13-4 so that it is only three icons wide, but twice as high, as in Figure 13-5. The first three icons from the box appear; the second three are obscured.* Notice the horizontal scrollbar at the bottom of the window, indicating that the other three icons are still to the right of these and thus not viewable in the resized box. If you place the pointer on the scrollbar, hold down the first button and drag the scrollbar to the right, the hidden icons will be revealed.

In order to rearrange the icons to better fill the new shape box, use the Pack Icons menu item. Figure 13-5 shows the icon box after you've selected Pack Icons.

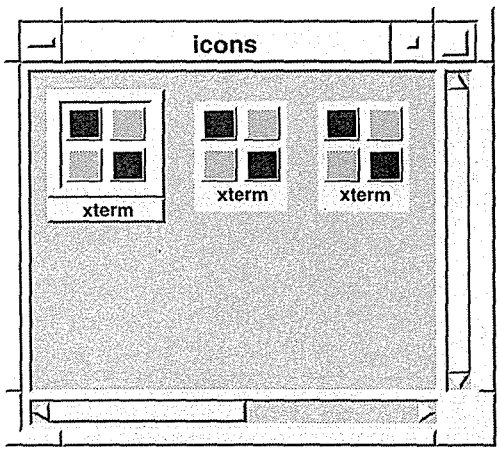

Figure 13-5. In the resized icon box, only three icons are visible

*When you resize the icon box, you'll notice the resize action has a tendency to jump the width or height of an icon at a time. *mwm* only allows the box to be resized exactly to fit a number of icons wide and a number high, though there are no obvious limitations as to the numbers. Basically, you can have an icon box of any size, even one icon high and wide, and display the other icons using the scrollbars. As you resize the box, the small rectangular window in the center of the screen assists you: it shows the dimensions in the number of icons wide by the number of icons high.

If you want to reorganize icons in the box yourself, without Pack Icons, this is also possible. You can actually move icons into adjacent empty slots using the pointer. Just hold down the first pointer button on the icon and drag it into the next slot. If you first make the icon box larger, so that there are several empty spaces, you'll find you can radically reorganize icons. Once you've arranged them as you like, you resize the box to fit the icons—or perhaps make it even smaller and view the obscured icons using the scrollbars.

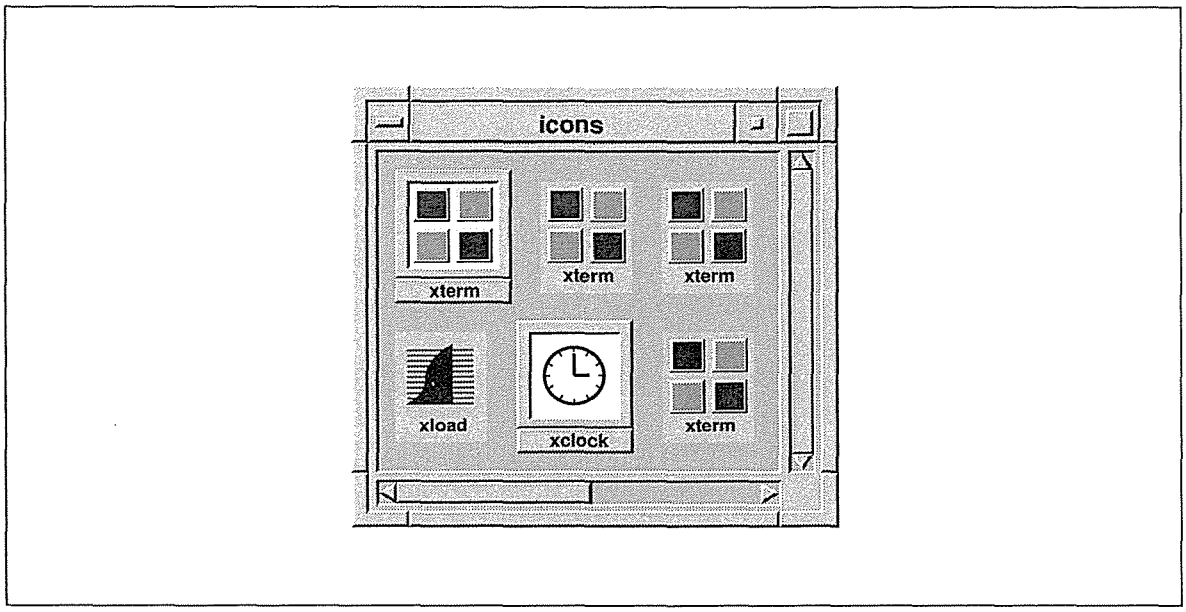

Figure 13-6. Pack Icons menu item rearranges icons in resized box

Keep in mind that the next time you log in, the icon box will be brought up at its default size. To specify alternate dimensions, set the variable `iconBoxGeometry` in your *.Xresources* file. For example, if you want an icon box three icons wide by two icons high, use the specification:

```
Mwm*iconBoxGeometry:  3x2+0-0
```

which creates a box of the desired size in the lower-left corner of the display. (This is the default location; you could omit the `+0-0` from the geometry string and get the same result.)

The following specification creates an icon box four icons wide by three icons high in the lower-right corner of the display:

```
Mwm*iconBoxGeometry:  4x3-0-0
```

14

Setup Clients

This chapter describes three useful setup clients that can be used to customize the appearance and operation of your display, and the operation of your keyboard and pointer.

In This Chapter:

14
Setup Clients

This chapter discusses how to set up certain features of your working environment, using these clients:

xset	To set certain characteristics of the keyboard, pointer, and display.
xsetroot	To set root window characteristics.
xmodmap	To change pointer and key mappings.

When Should I Set Preferences?

First, let's make one thing clear: you may never *have to* specify any user preferences at all. The default settings for your system may be sufficient for your needs.

In addition, some of the customizations you *can* make to your environment are purely aesthetic. The *xsetroot* client allows you to change what your root window looks like and what pointer shape is displayed on the root window. It has nothing to do with how X works.

On the other hand, both *xset* and *xmodmap* are primarily intended to control how your environment operates. You probably don't think about many of the features you can control using *xset*—features like pointer speed and how loud the keyboard bell is. *xset* allows you to fine tune fairly subtle aspects of your working environment, but in most cases, the system defaults will probably be sufficient. Perhaps more importantly, *xset* allows you to specify the font path—the directories in which the X server searches for fonts called by a client.

xmodmap is by far the most complicated and confusing of the setup clients. It allows you to change keyboard and pointer button "mappings," i.e., what a key or pointer button is assigned to do. For example, you might map a Delete function to the key marked "Backspace" on your keyboard. Chances are you'll want the so-called "Meta" function, which is used in many *mwm* window manager operations, to be mapped to a convenient key. (It may already be.) A left-handed person might want the rightmost button on the pointer to function as "pointer button one." (Generally, the leftmost button is pointer button one by default).

As you might imagine, pointer mappings are fairly simple. Most pointers have two or three physical buttons and the only thing you can change is their "logical" order, i.e., which is considered "first," etc. By contrast, some key mappings can seem like complex equations. Per-

haps the most confusing mappings involve the so-called "modifier" key functions (Control, Caps Lock, Meta, etc.), introduced in Chapter 4, *More about the mwm Window Manager.*

It's very possible that you may never have to specify any key or pointer customizations. The default assignments may be fine. If not, you can reassign functions using *xmodmap*. Later in this chapter, we discuss some of the issues you should be aware of before assigning key and pointer button functions and also show some typical mappings using *xmodmap*.

Although you can run setup commands at any time, as a general rule it's a good idea to put *xset* and *xsetroot* in your login session script. (This file is often called *.xinitrc* or *.xsession*. See Appendix A, *Managing Your Environment*, for details.) *xmodmap* is something of a special case. We'll come back to that.

The reasons for running *xset* and *xsetroot* at startup vary somewhat based on the display and server—but don't change the overall strategy. In most environments, user preferences are reset to the system defaults when you log out. It makes sense to specify your preferences in a session file that executes when you log in again.

In other cases, whatever preferences have been set for the server running the particular display (say an X terminal) will be carried over to the next login session on that display. (Some X terminals can be configured to retain settings between logins.) Hypothetically, another user could log in at your X terminal and get your preferences! If you use one display exclusively, this "inheritance" might seem more like an advantage than a problem. Keep in mind, however, that if the X server is restarted for some reason (and that will happen sooner or later), your preferences will be cleared and you'll have to run your setup commands next time you log in anyway. The simplest way to avoid problems like these is to run *xset* and *xsetroot* at startup.

Most of the settings controlled using *xset* and *xsetroot* will work on any display, with any server. Virtually the worst thing that can happen is that a server won't support a particular option and your command will simply be ignored. *xmodmap* is a special case because keyboards differ and thus many key assignments are not portable. If you always work at the same terminal, running *xmodmap* at startup should be no problem. If you're inclined to log in at different types of terminals, *xmodmap* can create problems. You should have a better idea of the possible complications after reading the section on *xmodmap* later in this chapter.

Now you might want to glance over the next few pages to get an idea of the types of features you can control using *xset* and *xsetroot*. Most of them are simple to specify and they can enhance your working environment. If you have problems with how the keyboard or pointer works (or think you'd like to improve it), read the section on *xmodmap*.

Setting Display and Keyboard Preferences: xset

You can specify many different "behaviors" of the display, pointer, and keyboard using the *xset* client. *xset* takes a variety of options that allow you to fine tune many features of your environment, including:

- The font search path
- The keyboard bell (margin bell)
- The keyclick (the noise each key makes when you type it)
- Keyboard autorepeat (the feature that causes a key to be typed multiple times if you hold it down)
- Pointer speed
- Screen saver

The command-line syntax for *xset* can be confusing. Some *xset* options are followed by `on` or `off` to set or unset the particular feature. Some options take a preceding dash to indicate that a feature be disabled, while the use of the option alone (without a dash) indicates that the feature be enabled. This can be confusing to users accustomed to seeing a dash as an introductory symbol on all options (as is the case with other UNIX and X programs).

Although you can run *xset* at any time, as a general rule it's a good idea to put it in your login session script. (This file is often called *.xinitrc* or *.xsession*. See Appendix A, *Managing Your Environment*, for details.)

The following sections describe the various options used to set pointer, keyboard, and screen features. Keep in mind that not all X implementations are guaranteed to honor all of these options.

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.

Volume as a percentage of the maximum is fairly easy to understand. A specification of `70` means that the volume will be "turned up" 70% of the way. The second and third arguments—pitch in hertz and duration in milliseconds—probably don't mean much to most users. One hundred milliseconds seems like a reasonable length beep. I don't notice much difference in pitch on my Sun workstation. I use the following settings:

```
% xset b 50 1000 100
```

This command sets the volume of the keyboard bell to 50 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`, the system default for volume is used. (Pitch and duration retain their previous settings.)

The command `xset b off` resets the volume to zero. (Again, pitch and duration retain their previous settings, but since the volume is turned all the way down, the bell is effectively disabled.) You can also turn off the bell by explicitly setting the volume parameter to 0 (`xset b 0`) or by using the `-b` option.

Bug Compatibility Mode (Release 4)

Some Release 3 clients were written to work with "features" of the R3 server, which could more accurately be called bugs. (Many of these bugs were eliminated in Release 4.) In order to allow certain R3 clients to work under R4, the R4 server has a bug compatibility mode that can be enabled using *xset*.

Bug compatibility mode is particularly important if you're running the R3 version of *xterm* under R4. Applications based on Version 1.0 of OSF/Motif (including Version 1.0 of *mwm*) also require that bug compatibility mode be enabled.

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

Keyclick Volume

The `c` option sets the volume of the keyboard's keyclick—a sound generated by the server when you type each key (not to be confused with the noise the physical key makes). To specify a particular level of keyclick, use the option:

```
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. Note also that in some cases, the keyclick cannot be turned on.

Enabling or Disabling Autorepeat

The `r` option controls the keyboard's autorepeat feature. Autorepeat causes a keystroke to be repeated over and over when the key is held down. (Multiple events are produced.) 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.

Changing or Rehashing the Font Path

As discussed in Chapter 6, *Font Specification*, by default the X server looks for fonts in four subdirectories of */usr/lib/X11/fonts*: *misc*, *Speedo*, *75dpi*, and *100dpi*.

You change the font path using *xset* with the `fp` (font path) option. See Chapter 6 for some examples.

Note that the `fp` option with the `rehash` parameter causes the server to reread the *fonts.dir* and *fonts.alias* files 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 (also discussed in Chapter 6). You also have to do this if you edit a *fonts.dir* file. See Volume Eight, *X Window System Administrator's Guide*, for more information.

Keyboard LEDs

The `led` option controls the enabling or disabling of one or all 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.

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 fraction, with the numerator and denominator 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 the acceleration parameter and then specify the threshold you want:

```
% xset m default 20
```

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 screen saver 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).

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.

Help with xset Options

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

Setting Root Window Characteristics: xsetroot

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 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, we suggest you run it from a startup shell script, as described in Appendix A, *Managing Your Environment*. See "When Should I Set Preferences?" earlier in this chapter for the reasoning behind this suggestion.

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.

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 standard bitmaps (generally found in the directory */usr/include/X11/bitmaps*) or make your own bitmap files using the *bitmap* client (see Chapter 7, *Graphics Utilities*).

For example, the command:

```
% xsetroot -bitmap /home/paula/gumby -fg red -bg blue
```

fills the root window with a tiling of the bitmap */home/paula/gumby* (a virtual army of Gumbys!), using the colors red 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. You can use a name from a color name database or a numeric color specification.

The command:

```
% xsetroot -solid lightblue
```

sets the color of the root window to light blue.* See Chapter 12, *Specifying Color*, for more about color possibilities.

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 a color name database, or as numeric values. See Chapter 12, *Specifying Color* for more instructions on how to specify color.

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

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

*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.

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.

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
```

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 (as well as pictures of them) in Appendix D, *Standard Cursors*. 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
```

The 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.*

For the *cursorfile*, you can use any of the standard bitmaps (generally found in */usr/include/X11/bitmaps*) or you can make your own with the *bitmap* client (see Chapter 7, *Graphics Utilities*).

Every standard cursor has an associated mask. To get an idea of what masks look like, display the cursor font using the command:

```
% 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
```

---

*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.

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.

†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*.

Then edit the *gumby.mask* file using the *bitmap* client, setting all squares inside the Gumby. (You can use the *bitmap* Flood Fill command 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 7, *Graphics Utilities*, for more information on using *bitmap*.

Modifier Key and Pointer Customization: xmodmap

Mapping keys is one of the more confusing tasks you might find the need to accomplish. When does key mapping become an issue? Here's a typical case. Say the key labeled "Control" is in a very awkward position for you and you have to use it all the time. Another key (labeled "Option") is in a very convenient position on the keyboard and you never use it. You can assign (or map) the Control function to the physical key labeled "Option" using *xmodmap*.

The *xmodmap* client is generally used to 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. (Basically, *xmodmap* can be used to specify what character is generated when you press a key or what action happens when you press a pointer button. You'll probably use it more often to map modifier key functions.)

As described in Chapter 3, *Working in the X Environment*, 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 confusion 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
- Control
- Mod1 (Meta in *mwm*)
- 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.

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, to 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.

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.

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 `-pke` 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*.

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 2 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
mod3
mod4
mod5
```

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*.)

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

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 `-pke` 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. (The "e" in `-pke` refers to "expression." This option specifies that each line in the map will be in the form of an expression that can in turn be supplied to *xmodmap*—to recover the original settings, if necessary.) Example 14-1 shows a portion of a typical keymap table returned by `xmodmap -pke`, for a Sun-3 keyboard.

Example 14-1. Partial keymap table with valid xmodmap expressions

```
keycode 109 = C
keycode 110 = V
keycode 111 = B
keycode 112 = N
keycode 113 = M
keycode 114 = comma less
keycode 115 = period greater
keycode 116 = slash question
keycode 117 = Shift_R
keycode 118 = Linefeed
keycode 119 = F33
keycode 120 = Down F34
keycode 121 = F35
          .
          .
          .
keycode 126 = Caps_Lock
keycode 127 = Meta_L
keycode 128 = space
keycode 129 = Meta_R
```

As you can see, the keymap table lists regular keyboard keys (C, V, comma, slash, space, etc.) and function/numeric keypad keys (F33, Down <Arrow>, F35, etc.), as well as modifier keys (Caps_Lock, Meta_L and Meta_R). Some keys generate two keysyms, the first when you press the key alone, the second when you hold Shift and then press the key. For example, the key with keycode 115 can generate a period (.) or (with Shift) the greater than symbol (>):

```
keycode 115 = period greater
```

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 -pke > keytable
```

You can recover the original mappings by supplying the relevant lines from the keymap to *xmodmap* using the `-e` option (explained later in this chapter).

The keysyms recognized by your server are a subset of a far greater number of keysyms recognized internationally. The file *keysym.h* (generally in the directory */usr/include/X11*) lists the keysym *families* that are enabled for your server. The file *keysymdef.h* (also generally in the directory */usr/include/X11*) 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.

Matching Keysyms with Physical Keys Using xev

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

To use *xev*, enter the command:

```
% xev
```

in an *xterm* window, and then use the pointer to place the *xev* window, as in Figure 14-1.

**xev* is a Release 3 standard client. Since Release 4, it has lived in the *demos* directory. If an executable version does not exist on your system, ask your system administrator.

If you cannot use *xev*, you must rely on the keymap table and a little deductive reasoning. Since certain *mwm* functions have keyboard shortcuts involving the Meta key, testing these shortcuts should help you locate this key. See Chapter 4, *More about the mwm Window Manager*, for more information.

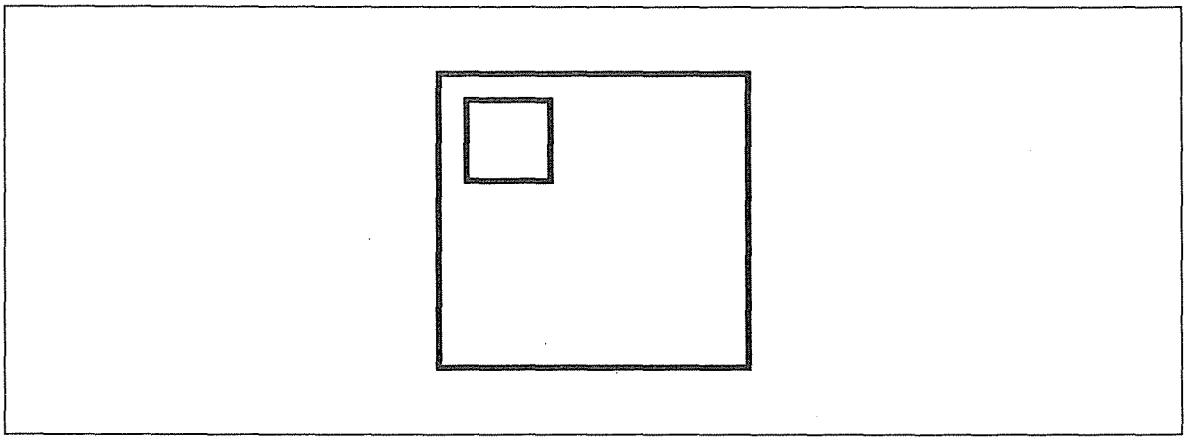

Figure 14-1. xev window

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. (This value cannot be supplied as a keysym argument to *xmodmap*.)

To find the Meta key, type a few likely keys in the *xev* window. Type Control-C in the window from which you invoked *xev* to terminate the program. (If you ran *xev* in the background, you'll have to kill the *xev* window. See Chapter 8, *Other Clients*, for ways to kill a client window.)

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*.

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.

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

`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.

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.

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*. (The section "When Do I Set Preferences?" discusses some issues regarding when to run *xmodmap*.) The current 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 map.)

First, the *xmodmap* client 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. For 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 -pke | grep Delete
```

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

```
keycode 50 = Delete
keycode 73 = 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 -pke` 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.

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 14-2.

```
There are 3 pointer buttons defined.

    Physical        Button
     Button          Code
        1              1
        2              2
        3              3
```

Figure 14-2. 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 leftmost button, designed to be pressed by the right index finger. The *xmodmap* client allows you to reassign logical buttons—typically so that the pointer can be more easily used with the left hand.*

There are two relevant *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 = n1 n2 n3`

Sets the first, second, and third physical buttons to the button codes *n1*, *n2*, and *n3*.

`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 (that is, generate button code 1). To configure the pointer for a southpaw:

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

Then if you 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
```

*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.

You can then push the first logical button (button code 1) with the index finger of your left hand.

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

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

Part Three:

Client Reference Pages

This part of the guide provides UNIX-style "man-pages" for each of the standard X programs, as well as the mwm *window manager. These pages are arranged alphabetically for ease of reference, and they contain detailed information (such as all options to a program) that is not covered in other parts of this guide.*

The following reference pages appear in this section:

Intro	xdm
X	xdpr
Xserver	xdpyinfo
appres	xedit
bdftopcf	xev
bdftosnf	xfd
bitmap	xfontsel
editres	xhost
fs	xinit
fsinfo	xkill
fslsfonts	xload
fstobdf	xlogo
listres	xlsatoms
mkfontdir	xlsclients
mwm	xlsfonts
oclock	xlswins
resize	xmag
sessreg	xman
showfont	xmh
showrgb	xmodmap
showsnf	xpr
viewres	xprop
xauth	xrdb
xbiff	xrefresh
xcalc	xsccd
xclipboard	xset
xclock	xsetroot
xcmsdb	xstdcmap
xcol	xterm
xcoloredit	xtici
xconsole	xwd
xcrtca	xwininfo
xditview	xwud

Name

Intro – overview of reference page format.

Syntax

This section describes the command-line syntax for invoking the client. Anything in **bold** type should be typed exactly as shown. Items in *italics* are parameters that should be replaced by actual values when you enter the command. Anything enclosed in brackets is optional. For example:

bitmap [*options*] *filename*

means to type the command **bitmap** followed by zero or more options (from the list of options on the reference page), followed by the name of the file containing the bitmap to be edited.

Description

This section explains the operation of the client. In some cases, additional descriptive sections appear later on in the reference page.

Options

This section lists available command-line options. In some cases, reference is made to "all of the standard X Toolkit command-line options." These X Toolkit options are listed in Chapter 10 of this guide, as well as in the first reference page in this section, which is simply labeled *X*.

Resources

This section lists the resource variable names that can be specified in an *.Xresources* or other resource file. In some cases, reference is made to "all the core resource names and classes." A list of the core names and classes appears in Appendix G, *Widget Resources*. Syntax rules and examples appear in Chapter 11, *Setting Resources*. For complete information, see Volume Four, *X Toolkit Intrinsics Programming Manual*.

Widget Hierarchy

Applications written with the X Toolkit are comprised of widgets, which are predefined user interface components or objects. Typical widgets create graphical features such as menus, command buttons, dialog boxes, and scrollbars. Applications composed of widgets are always window-based (such as *xterm*, *xclock*, and *xman*). X also provides clients that are not window-based (such as *xlsfonts*, *xwininfo*, and *xlsclients*) and thus do not use widgets.

If present on a reference page, the section "Widget Hierarchy" diagrams the relationship of the widgets within the application. The widget hierarchy is significant in specifying client resources. Most Toolkit clients accept both application-specific resources (listed in the "Resources" section) and resources for the component widgets. Appendix G lists the user-settable resources for the Athena widgets and explains the somewhat complicated mechanisms by which resources are interpreted.

Files

If present, this section lists the system and/or application-specific files relevant to the application.

Environment

If present, this section lists shell environment variables used by the client. This section does not list the DISPLAY and XENVIRONMENT variables, which are used by all clients. These variables are used as follows:

DISPLAY

To get the default display name (specifically, the host, server/display, and screen). The DISPLAY variable typically has the form:

hostname`:`*server*`.`*screen*

(for example, `isla:0.0`). See *X* for more information about display name syntax.

XENVIRONMENT

To get the name of a resource file containing host-specific resources. If this variable is not set, the resource manager will look for a file called *.Xdefaults-**hostname*** (where ***hostname*** is the name of a particular host) in the user's home directory. See the *X* reference page for more information.

See Also

This section lists other reference pages in Part Three of this guide that may also be of interest. Note that versions of these pages may have been installed in the usual online manual page directories, and may be available via the UNIX *man*(1) command. References such as *stat*(2) can be found in the standard UNIX documentation. This section may also include references to documentation on Xlib, the X Toolkit, various widgets, etc.

Bugs

If present, this section lists areas in which the author of the program thinks it could be improved. In a few instances, we've listed additional bugs we've noted.

Author

The authors of the program and (generally) of the reference page as well. Most of the reference pages are subject to the copyright provisions in the "Copyright" section of the X reference page. Where appropriate, additional copyrights are noted on individual pages.

Note, however, that those portions of this document that are based on the original X11 documentation and other source materials have been revised, and that all such revisions are copyright © 1987, 1988, 1989, 1990, 1991, 1992 O'Reilly & Associates, Inc. Inasmuch as the proprietary revisions can't be separated from the freely copyable MIT source material, the net result is that copying of this document is not allowed. Sorry for the doublespeak!

Name

X – a portable, network-transparent window system.

Description

X is a network-transparent window system developed at MIT that runs on a wide range of computing and graphics machines. It should be relatively straightforward to build the MIT software distribution on most ANSI C and POSIX compliant systems. Commercial implementations are also available for a wide range of platforms.

The X Consortium requests that the following names be used when referring to this software:

X
X Window System
X Version 11
X Window System, Version 11
X11

The name "X Windows" should not be used. *X Window System* is a trademark of the Massachusetts Institute of Technology.

X Window System servers run on computers with bitmap displays. The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can also be run transparently from other machines, including machines with different architectures and operating systems.

X supports overlapping hierarchical subwindows, and text and graphics operations, on both monochrome and color displays. For a full explanation of the functions that are available, see Volume One, *Xlib Programming Manual*, and Volume Two, *Xlib Reference Manual*.

The number of programs that use X is quite large. Programs provided in the core MIT distribution include: a terminal emulator (*xterm*), a window manager (*twm*), a display manager (*xdm*), a console redirect program (*xconsole*), mail managing utilities (*xmh* and *xbiff*), a manual page browser (*xman*), a bitmap editor (*bitmap*), a resource editor (*editres*), a ditroff previewer (*xditview*), access control programs (*xauth* and *xhost*), user preference setting programs (*xrdb*, *xcmsdb*, *xset*, *xsetroot*, *xstdcmap*, and *xmodmap*), a load monitor (*xload*), clocks (*oclock* and *xclock*), a font displayer (*xfd*), utilities for listing information about fonts, windows, and displays (*xlsfonts*, *xfontsel*, *xstdcmap*, *xwininfo*, *xdpyinfo*, *xlsclients*, and *xprop*), a diagnostic for seeing what events are generated and when (*xev*), screen image manipulation utilities (*xwd*, *xwud*, *xpr*, and *xmag*), and various demos (*xeyes*, *ico*, *xllperf*, *xgc*, etc.)

Many other utilities, window managers, games, toolkits, and so on, are included as user-contributed software in the MIT distribution, or are available using anonymous *ftp* on the Internet. For more information, see the Preface of this guide, Volume Eight, *X Window System Administrator's Guide*; and Volume One, *Xlib Programming Manual*.

Starting Up

There are two ways of starting the X server and an initial set of client applications. The particular method used depends on which operating system you are running and on whether or not you use other window systems in addition to X. The methods are:

xdm (the X Display Manager)

If you want to have X running on your display at all times, your site administrator can set up your machine to use the X Display Manager *xdm*. This program is typically started by the system at boot time and takes care of keeping the server running and getting users logged in. If you are running *xdm*, you will see a window on the screen welcoming you to the system and asking for your username and password. Simply type them in as you would at a normal terminal, pressing the Return key after each. If you make a mistake, *xdm* will display an error message and ask you to try again. After you have successfully logged in, *xdm* will start up your X environment. By default, if you have an executable file named *.xsession* in your home directory, *xdm* will treat it as a program (or shell script) to be run to start up your initial clients (such as terminal emulators, clocks, a window manager, user settings for things like the background, the speed of the pointer, etc.). Your site administrator can provide details.

xinit (run manually from the shell)

Sites that support more than one window system might choose to use the *xinit* program for starting X manually. If this is true for your machine, your site administrator will probably have provided a program named "x11", "startx", or "xstart" that will do site-specific initialization in a nice way (such as loading convenient default resources, running a window manager, displaying a clock, and starting several terminal emulators). If not, you can build such a script using the *xinit* program. This utility simply runs one user-specified program to start the server, runs another to start up any desired clients, and then waits for either to finish. Since either or both of the user-specified programs may be a shell script, this gives substantial flexibility at the expense of a nice interface. For this reason, *xinit* is not intended for end users.

Display Names

From the user's perspective, every X server has a *displayname* of the form:

host`:`*server*`.`*screen*

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

host The host name of the physical display. If the *host* name is not given, the most efficient way of communicating to a server on the same machine will be used.

server

The server (or display) number. The phrase "display" is usually used to refer to a collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most workstations have only one keyboard, and therefore only one display. Larger, multi-user systems, however, will frequently have several displays so that more than

one person at a time can be doing graphics work. To avoid confusion, each display on a machine is assigned a `server` number (beginning at 0) when the X server for that display is started. The `server` number must always be given in a display name. In this guide, the `server` number is also referred to as the `display` number (referring to the phrase *display server*).

`screen`

The *`screen`* number. Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a *`screen`* number (beginning at 0) when the X server for that display is started. If the *`screen`* number is not given, then screen 0 will be used.

On POSIX systems, the default display name is stored in your DISPLAY environment variable. This variable is set automatically by the *xterm* terminal emulator. However, when you log into another machine on a network, you'll need to set DISPLAY by hand to point to your display. For example:

```
% setenv DISPLAY myws:0                (C Shell)
$ DISPLAY=myws:0; export DISPLAY       (Bourne Shell)
```

The *xon* script can be used to start an X program on a remote machine. It automatically sets the DISPLAY variable correctly. Finally, most X programs accept a command-line option of `-display` *`displayname`* to temporarily override the contents of DISPLAY. This is most commonly used to pop windows on another person's screen or as part of a "remote shell" command to start an xterm pointing back to your display. For example:

```
% xeyes -display joesws:0 -geometry 1000x1000+0+0
% rsh big xterm -display myws:0 -ls </dev/null &
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, the *`host`* name part of the display name is used to determine the type of channel (also called a transport layer) to be used. X servers generally support the following types of connections:

local The *`host`* part of the display name should be the empty string. For example: `:0`, `:1`, and `:0.1`. The most efficient local transport will be chosen.

TCP/IP The *`host`* part of the display name should be the server machine's IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example: `expo.lcs.mit.edu:0`, `expo:0`, `18.30.0.212:0`, `bigmachine:1`, and `hydra:0.1`.

DECnet

The *`host`* part of the display name should be the server machine's nodename followed by two colons instead of one. For example: `myws::0`, `big::1`, and `hydra::0.1`.

Access Control

An X server can use several types of access control. Mechanisms provided in Release 5 are:

```
Host Access Simple host-based access control.
MIT-MAGIC-COOKIE-1  Shared plain-text "cookies".
XDM-AUTHORIZATION-1 Secure DES based private-keys.
SUN-DES-1   Based on Sun's secure rpc system.
```

xdm initializes access control for the server, and also places authorization information in a file accessible to the user and the server. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients that are explicitly authorized can connect to the display. When you add entries to the host list (with *xhost*), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file from which Xlib extracts authorization data can be specified with the environment variable XAUTHORITY, and defaults to the file *.Xauthority* in the home directory. *xdm* uses *$HOME/.Xauthority* and will create it or merge in authorization records if it already exists when a user logs in.

If you use several machines, and share a common home directory across all of the machines by means of a network file system, then you never really have to worry about authorization files; the system should work correctly by default. Otherwise, as the authorization files are machine-independent, you can simply copy the files to share them. To manage authorization files, use *xauth*. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you login, if the remote machine does not share a common home directory with your local machine. Note that authorization information transmitted "in the clear" through a network file system or using *ftp* or *rcp* can be "stolen" by a network eavesdropper, and as such may enable unauthorized access. In many environments this level of security is not a concern, but if it is, you need to know the exact semantics of the particular authorization data to know if this is actually a problem.

For more information on access control, see the *Xsecurity* manual page.

For more information on access control, see Appendix A, *Managing Your Environment*, Volume One, *Xlib Programming Manual*, and the *Xsecurity* reference page in the MIT source distribution.

Geometry Specifications

One of the advantages of using window systems instead of hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running (described below), most X programs accept a command-line argument of the form `-geometry` *`widthxheight±xoff±yoff`* (where *`width`*, *`height`*, *`xoff`*, and *`yoff`* are numbers) for specifying a preferred size and location for this application's main window.

The *`width`* and *`height`* parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The *`xoff`* and *`yoff`* parts are measured in

pixels and are used to specify the distance of the window from the left or right and top and bottom edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window. The x offset may be specified in the following ways:

+xoff The left edge of the window is to be placed *xoff* pixels in from the left edge of the screen (i.e., the x coordinate of the window's origin will be *xoff*). *xoff* may be negative, in which case the window's left edge will be off the screen.

-xoff The right edge of the window is to be placed *xoff* pixels in from the right edge of the screen. *xoff* may be negative, in which case the window's right edge will be off the screen.

The y offset has similar meanings:

+yoff The top edge of the window is to be *yoff* pixels below the top edge of the screen (i.e., the y coordinate of the window's origin will be *yoff*). *yoff* may be negative, in which case the window's top edge will be off the screen.

-yoff The bottom edge of the window is to be *yoff* pixels above the bottom edge of the screen. *yoff* may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either *xoff* or *yoff*, both must be present. Windows can be placed in the four corners of the screen using the following specifications:

+0+0 The upper-left corner.

-0+0 The upper-right corner.

-0-0 The lower-right corner.

+0-0 The lower-left corner.

In the following examples, a terminal emulator will be placed in roughly the center of the screen and a load average monitor, mailbox, and clock will be placed in the upper-right corner:

```
% xterm -fn 6x10 -geometry 80x24+30+200 &
% xclock -geometry 48x48-0+0 &
% xload -geometry 48x48-96+0 &
% xbiff -geometry 48x48-48+0 &
```

Window Managers

The layout of windows on the screen is controlled by special programs called *window managers*. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The MIT distribution comes with a window manager named *twm*,

which supports overlapping windows, popup menus, point-and-click or click-to-type input models, titlebars, nice icons (and an icon manager for those who don't like separate icon windows).

See the user-contributed software in the MIT distribution for other popular window managers.

Font Names

Collections of characters for displaying text and symbols in X are known as *fonts*. A font typically contains images that share a common appearance and look nice together (for example, a single size, boldness, slant, and character set). Similarly, collections of fonts that are based on a common type face (the variations are usually called roman, bold, italic, bold italic, oblique, and bold oblique) are called *families*.

Fonts come in various sizes. The X server supports *scalable* fonts, meaning it is possible to create a font of arbitrary size from a single source for the font. The server supports scaling from *outline* fonts and *bitmap* fonts. Scaling from outline fonts usually produces significantly better results than scaling from bitmap fonts.

An X server can obtain fonts from individual files stored in directories in the file system, or from one or more font servers, or from a mixture of directories and font servers. The list of places the server looks when trying to find a font is controlled by its *font path*. Although most installations will choose to have the server start up with all of the commonly used font directories in the font path, the font path can be changed at any time with the *xset* program. However, it is important to remember that the directory names are on the *server*'s machine, not on the application's.

The default font path for the X server contains four directories:

/usr/lib/X11/fonts/misc

This directory contains many miscellaneous bitmap fonts that are useful on all systems. It contains a family of generic fixed-width fonts, a family of fixed-width fonts from Dale Schumacher, several Kana fonts from Sony Corporation, two JIS Kanji fonts, two Hangul fonts from Daewoo Electronics, two Hebrew fonts from Joseph Friedman, the standard cursor font, two cursor fonts from Digital Equipment Corporation, and cursor and glyph fonts from Sun Microsystems. It also has various font name aliases for the fonts, including *fixed* and *variable*.

/usr/lib/X11/fonts/Speedo

This directory contains outline fonts for Bitstream's Speedo rasterizer. A single font face, in normal, bold, italic, and bold italic, is provided, contributed by Bitstream, Inc.

/usr/lib/X11/fonts/75dpi

This directory contains 75 dots per inch display bitmap fonts contributed by Adobe Systems, Inc.; Digital Equipment Corporation, Bitstream, Inc.; Bigelow and Holmes, and Sun Microsystems, Inc. An integrated selection of sizes, styles, and weights are provided for each family.

/usr/lib/X11/fonts/100dpi

This directory contains 100 dots per inch versions of some of the fonts in the *75dpi* directory.

Font databases are created by running the *mkfontdir* program in the directory containing the source or compiled versions of the fonts. Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the *xset* program. For example, to add a font to a private directory, the following commands could be used:

```
% cp newfonts.pcf ~/myfonts
% mkfontdir ~/myfonts
% xset fp rehash
```

The *xfontsel* and *xlsfonts* programs can be used to browse through the fonts available on a server. Font names tend to be fairly long as they contain all of the information needed to uniquely identify individual fonts. However, the X server supports wildcarding of font names, so the full specification:

```
-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1
```

might be abbreviated as:

```
*-courier-medium-r-normal--*-100-*
```

Because the shell also has special meanings for * and ?, wildcarded font names should be quoted:

```
% xlsfonts -fn '*-courier-medium-r-normal--*-100-*'
```

The *xlsfonts* program can be used to list all of the fonts that match a given pattern. With no arguments, it lists all available fonts. This will usually list the same font at many different sizes. To see just the base scalable font names, try using one of the following patterns:

--*-*-*-*-*-0-0-0-0-*-0-*-*
--*-*-*-*-*-0-0-75-75-*-0-*-*
--*-*-*-*-*-0-0-100-100-*-0-*-*

To convert one of the resulting names into a font at a specific size, replace one of the first two zeros with a nonzero value. The field containing the first zero is for the pixel size; replace it with a specific height in pixels to name a font at that size. Alternatively, the field containing the second zero is for the point size; replace it with a specific size in decipoints (there are 722.7 decipoints to the inch) to name a font at that size. The last zero is an average width field, measured in tenths of pixels; some servers will anamorphically scale if this value is specified.

If more than one font in a given directory in the font path matches a wildcarded font name, the choice of which particular font to return is left to the server. However, if fonts from more than one directory match a name, the returned font will always be from the first such directory in the font path. The example given above will match fonts in both the *75dpi* and *100dpi* directories;

if the *75dpi* directory is ahead of the *100dpi* directory in the font path, the smaller version of the font will be used.

Font Server Names

One of the following forms can be used to name a font server that accepts TCP connections:

tcp/*hostname*:*port*
tcp/*hostname*:*port*/*cataloguelist*

The *hostname* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *port* is the decimal TCP port on which the font server is listening for connections. The *cataloguelist* specifies a list of catalogue names, with '+' as a separator.

Examples: *tcp/expo.lcs.mit.edu:7000*, *tcp/18.30.0.212:7001/all*.

One of the following forms can be used to name a font server that accepts DECnet connections:

decnet/*nodename*::font$*objname*
decnet/*nodename*::font$*objname*/*cataloguelist*

The *nodename* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *objname* is a normal, case-insensitive DECnet object name. The *cataloguelist* specifies a list of catalogue names, with '+' as a separator.

Examples: *DECnet/SRVNOD::FONT$DEFAULT*, *decnet/44.70::font$special/symbo ls*.

Color Names

Most applications provide ways of tailoring (usually through resources or command-line arguments) the colors of various elements in the text and graphics they display.

X supports the use of abstract color names, for example, "red", "blue." A value for this abstract name is obtained by searching one or more color name databases. *Xlib* first searches zero or more client-side databases; the number, location, and content of these databases is implementation dependent. If the name is not found, the color is looked up in the X server's database. The text form of this database is commonly stored in the file */usr/lib/X11/rgb.txt*.

A color can be specified either by an abstract color name, or by a numerical color specification. The numerical specification can identify a color in either device-dependent (RGB) or device-independent terms. Color strings are case-insensitive meaning that *red*, *Red*, and *RED* all refer to the same color.

A numerical color specification consists of a color space name and a set of values in the following syntax:

<color_space_name>:*<value>*/.../*<value>*

An RGB Device specification is identified by the prefix "rgb:" and has the following syntax:

```
rgb:<red>/<green>/<blue>

    <red>, <green>, <blue> := h | hh | hhh | hhhh
    h := single hexadecimal digits
```

Note that *h* indicates the value scaled in 4 bits, *hh* the value scaled in 8 bits, *hhh* the value scaled in 12 bits, and *hhhh* the value scaled in 16 bits, respectively. These values are passed directly to the X server, and are assumed to be gamma corrected.

The eight primary colors can be represented as:

```
black                   rgb:0/0/0
red                     rgb:ffff/0/0
green                   rgb:0/ffff/0
blue                    rgb:0/0/ffff
yellow                  rgb:ffff/ffff/0
magenta                 rgb:ffff/0/ffff
cyan                    rgb:0/ffff/ffff
white                   rgb:ffff/ffff/ffff
```

For backward compatibility, an older syntax for RGB Device is supported, but its continued use is not encouraged. The syntax is an initial sharp sign character followed by a numeric specification, in one of the following formats:

```
#rgb                    (4 bits each)
#rrggbb                 (8 bits each)
#rrrgggbbb              (12 bits each)
#rrrrggggbbbb           (16 bits each)
```

The r, g, and b represent single hexadecimal digits indicating how much *red*, *green*, and *blue* should be displayed (zero being none and ffff being on full). Each field in the specification must have the same number of digits (e.g., *#rrgb* or *#gbb* are not allowed). When fewer than 16 bits each are specified, they represent the most significant bits of the value (unlike the "rgb:" syntax, in which values are scaled). For example, #3a7 is the same as #3000a0007000.

An RGB intensity specification is identified by the prefix "rgbi:" and has the following syntax:

```
rgbi:<red>/<green>/<blue>
```

The red, green, and blue are floating point values between 0.0 and 1.0, inclusive. They represent linear intensity values, with 1.0 indicating full intensity, 0.5 half intensity, and so on. These values will be gamma corrected by *Xlib* before being sent to the X server. The input format for these values is an optional sign, a string of numbers possibly containing a decimal point, and an optional exponent field containing an E or e followed by a possibly signed integer string.

The standard device-independent string specifications have the following syntax:

```
CIEXYZ:<X>/<Y>/<Z>          (none, 1, none)
CIEuvY:<u>/<v>/<Y>          (~.6, ~.6, 1)
CIExyY:<x>/<y>/<Y>          (~.75, ~.85, 1)
```

CIELab:<*L*>/<*a*>/<*b*>	(100, *none*, *none*)
CIELuv:<*L*>/<*u*>/<*v*>	(100, *none*, *none*)
TekHVC:<*H*>/<*V*>/<*C*>	(360, 100, 100)

All of the values (C, H, V, X, Y, Z, a, b, u, v, y, x) are floating point values. Some of the values are constrained to be between zero and some upper bound; the upper bounds are given in parentheses above. The syntax for these values is an optional '+' or '-' sign, a string of digits possibly containing a decimal point, and an optional exponent field consisting of an 'E' or 'e' followed by an optional '+' or '-' followed by a string of digits.

For more information on device independent color, see Volume Two, *Xlib Reference Manual*.

Keys

The X keyboard model is broken into two layers: server-specific codes (called *keycodes*), which represent the physical keys; and server-independent symbols (called *keysyms*), which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

modifier list

Some keys (such as Shift, Control, and Caps Lock) are known as *modifier keys* and are used to select different symbols that are attached to a single key (such as Shift-a generates a capital A, and CTRL-L generates a control character ^L). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is pressed or released, the server generates an *event* that contains the keycode of the indicated key as well as a mask that specifies which of the modifer keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift lock keys on the keyboard.

keymap table

Applications translate event keycodes and modifier masks into keysyms using a *keymap table* which contains one row for each keycode and one column for each of the modifiers. This table is initialized by the server to correspond to normal typewriter conventions. The exact semantics of how the table is interpreted to produce keysyms depends on the particular program, libraries, and language input method used, but the following conventions for the first four keysyms in each row are generally adhered to.

The first four elements of the list are split into two groups of keysyms. Group 1 contains the first and second keysyms; Group 2 contains the third and fourth keysyms. Within each group, if the first element is alphabetic and the the second element is the special keysym *NoSymbol*, then the group is treated as equivalent to a group in which the first element is the lowercase letter and the second element is the uppercase letter.

Switching between groups is controlled by the keysym named MODE SWITCH, by attaching that keysym to some key and attaching that key to any one of the modifiers Mod1 through Mod5. This modifier is called the "group modifier." Group 1 is used when the group modifier is off, and Group 2 is used when the group modifier is on.

Within a group, the modifier state determines which keysym to use. The first keysym is used when the Shift and Lock modifiers are off. The second keysym is used when the Shift modifier is on, when the Lock modifier is on and the second keysym is uppercase alphabetic, or when the Lock modifier is on and is interpreted as ShiftLock. Otherwise, when the Lock modifier is on and is interpreted as CapsLock, the state of the Shift modifier is applied first to select a keysym; but if that keysym is lowercase alphabetic, then the corresponding uppercase keysym is used instead.

Options

Most X programs attempt to use the same names for command-line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

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

Specifies the name of the display to use. *`host`* is the hostname of the physical display, *`server`* specifies the display server number, and *`screen`* specifies the screen number. Either or both of the *`host`* and *`screen`* elements to the display specification can be omitted. If *`host`* is omitted, the local display is assumed. If *`screen`* is omitted, screen 0 is assumed (and the period is unnecessary). The colon and (display) *`server`* are necessary in all cases.

For example, the following command creates an *xclock* window on screen 1 on server 0 on the display hardware named by *`your_node`*.

```
% xclock -display your_node:0.1
```

The `-display` option can be abbreviated as `-d`, unless the client accepts another option that begins with "d."

`-geometry` *`geometry`*

Specifies the initial size and location of the application window. The `-geometry` option can be (and often is) abbreviated to `-g`, unless the client accepts another option that begins with "g." The argument (*`geometry`*) is referred to as a "standard geometry string," and has the form *`width`*`x`*`height`*±*`xoff`*±*`yoff`*.

`-bg` *`color`*, `-background` *`color`*

Either option specifies the color to use for the window background.

`-bd` *`color`*, `-bordercolor` *`color`*

Either option specifies the color to use for the window border.

`-bw` *`pixels`*, `-borderwidth` *`pixels`*

Either option specifies the width in pixels of the window border.

`-fg` *`color`*, `-foreground` *`color`*

Either option specifies the color to use for text or graphics.

`-fn` *`font`*, `-font` *`font`*

Either option specifies the font to use for displaying text.

`-iconic`
: Indicates that the user would prefer that the application's windows initially not be visible, as if the windows had been immediately iconified by the user. Window managers may choose not to honor the application's request.

`-name` *app_name*
: Specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable filename.

`-rv, -reverse`
: Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually used only on monochrome displays.

`+rv`
: Indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

`-selectionTimeout` *milliseconds*
: Specifies the timeout in milliseconds within which two communicating applications must respond to one another for a selection request.

`-synchronous`
: Indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since Xlib normally buffers requests to the server, errors are not necessarily reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

`-title` *string*
: Specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.

`-xnllanguage` *language*[*_territory*][*.codeset*]
: Specifies the language, territory, and codeset for use in resolving resource and other filenames.

`-xrm` *resourcestring*
: Specifies a resource name and value to override any defaults. It is very useful for setting resources that don't have explicit command line arguments.

Resources

To make the tailoring of applications to personal preferences easier, X provides a mechanism for storing default values for program resources (e.g., background color, window title, etc.). Resources are specified as strings of the form:

*appname*subname*subsubname...: value*

and are read in from various places when an application is run. Program components are named in a hierarchical fashion, with each node in the hierarchy identified by a class and an instance name. At the top level is the class and instance name of the application itself.

By convention, the class name of the application is the same as the program name, but with the first letter capitalized (e.g., *Bitmap* or *Emacs*), although some programs that begin with the letter "x" also capitalize the second letter for historical reasons.

The precise syntax for resources is:

```
ResourceLine    =       Comment | IncludeFile | ResourceSpec | <empty line>
Comment         =       "!" {<any character except null or newline>}
IncludeFile     =       "#" WhiteSpace "include" WhiteSpace FileName WhiteSpace
FileName        =       <valid filename for operating system>
ResourceSpec    =       WhiteSpace ResourceName WhiteSpace ":" WhiteSpace Value
ResourceName    =       [Binding] {Component Binding} ComponentName
Binding         =       "." | "*"
WhiteSpace      =       {<space> | <horizontal tab>}
Component       =       "?" | ComponentName
ComponentName   =       NameChar {NameChar}
NameChar        =       "a"-"z" | "A"-"Z" | "0"-"9" | "_" | "-"
Value           =       {<any character except null or unescaped newline>}
```

Note that elements separated by a vertical bar (|) are alternatives. Elements enclosed in curly braces ({...}) indicate zero or more occurrences of the enclosed elements. Square brackets ([...]) indicate that the enclosed element is optional. Quotes ("...") are used around literal characters.

IncludeFile lines are interpreted by replacing the line with the contents of the specified file. The word "include" must be in lowercase. The filename is interpreted relative to the directory of the file in which the line occurs (for example, if the filename contains no directory or contains a relative directory specification).

If a ResourceName contains a contiguous sequence of two or more Binding characters, the sequence will be replaced with a single "." character if the sequence contains only "." characters; otherwise, the sequence will be replaced with a single "*" character.

A resource database never contains more than one entry for a given ResourceName. If a resource file contains multiple lines with the same ResourceName, the last line in the file is used.

Any whitespace character before or after the name or colon in a ResourceSpec is ignored. To allow a Value to begin with whitespace, the two-character sequence "*space*" (backslash followed by space) is recognized and replaced by a space character, and the two-character sequence "*tab*" (backslash followed by horizontal tab) is recognized and replaced by a horizontal tab character. To allow a Value to contain embedded newline characters, the two-character sequence "\n" is recognized and replaced by a newline character. To allow a Value to be broken across multiple lines in a text file, the two-character sequence "*newline*" (backslash followed by newline) is recognized and removed from the value. To allow a Value to contain arbitrary character codes, the four-character sequence "*nnn*", where each *n* is a digit character in

the range of "0"–"7", is recognized and replaced with a single byte that contains the octal value specified by the sequence. Finally, the two-character sequence "\\" is recognized and replaced with a single backslash.

When an application looks for the value of a resource, it specifies a complete path in the hierarchy, with both class and instance names. However, resource values are usually given with only partially specified names and classes, using pattern matching constructs. An asterisk (*) is a loose binding and is used to represent any number of intervening components, including none. A period (.) is a tight binding and is used to separate immediately adjacent components. A question mark (?) is used to match any single component name or class. A database entry cannot end in a loose binding; the final component (which cannot be "?") must be specified. The lookup algorithm searches the resource database for the entry that most closely matches (is most specific for) the full name and class being queried. When more than one database entry matches the full name and class, precedence rules are used to select just one.

The full name and class are scanned from left to right (from highest level in the hierarchy to lowest), one component at a time. At each level, the corresponding component and/or binding of each matching entry is determined, and these matching components and bindings are compared according to precedence rules. Each of the rules is applied at each level, before moving to the next level, until a rule selects a single entry over all others. The rules (in order of precedence) are:

1. An entry that contains a matching component (whether name, class, or "?") takes precedence over entries that elide the level (that is, entries that match the level in a loose binding).
2. An entry with a matching name takes precedence over both entries with a matching class and entries that match using "?". An entry with a matching class takes precedence over entries that match using "?".
3. An entry preceded by a tight binding takes precedence over entries preceded by a loose binding.

Programs based on the X Toolkit Intrinsics obtain resources from the following sources (other programs usually support some subset of these sources):

RESOURCE_MANAGER root window property
: Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window of the first screen using the *xrdb* program. This is frequently taken care of when the user starts up X through the display manager or *xinit*.

SCREEN_RESOURCES root window property
: Any resources specific to a given screen (e.g., colors) that should be available to clients on all machines, should be stored in the SCREEN_RESOURCES property on the root window of that screen. The *xrdb* program will sort resources automatically and place them in RESOURCE_MANAGER or SCREEN_RESOURCES, as appropriate.

application-specific files

Directories named by the environment variable XUSERFILESEARCHPATH or the environment variable XAPPLRESDIR, plus directories in a standard place (usually under */usr/lib/X11/*, but this can be overriden with the XFILESEARCHPATH environment variable) are searched for application-specific resource files. Files are generally named *Class*—for the class name of the application.

XAPPLRESDIR configuration files are actually loaded *before* the RESOURCE_MANAGER property, so that the property can override the values. See Volume Four, *X Toolkit Intrinsics Programming Manual*, for details.

XENVIRONMENT

Any user- and machine-specific resources may be specified by setting the XENVIRONMENT environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, a file named *$HOME/.Xdefaults-**hostname*** is looked for instead, where ***hostname*** is the name of the host where the application is executing.

`-xrm` *`resourcestring`*

Resources can also be specified from the command line. The *`resourcestring`* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of `-xrm` arguments may be given on the command line.

Program resources are organized into groups called *classes*, so that collections of individual resources (each of which is called an *instance*) can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an uppercase letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics will have at least the following resources:

`background` (class `Background`)

Specifies the color to use for the window background.

`borderWidth` (class `BorderWidth`)

Specifies the width in pixels of the window border.

`borderColor` (class `BorderColor`)

Specifies the color to use for the window border.

Most applications using the X Toolkit Intrinsics also have the resource `foreground` (class `Foreground`), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set `Background` and `Foreground` classes to particular defaults. Specific color instances, such as text cursors, can then be overridden without having to define all of the related resources. For example,

```
bitmap*Dashed:  off
XTerm*cursorColor:  gold
XTerm*multiScroll:  on
XTerm*jumpScroll:  on
XTerm*reverseWrap:  on
XTerm*curses:  on
XTerm*Font:  6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime: 500
XTerm*charClass:  33:48,37:48,45-47:48,64:48
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit:  on
XTerm*ttyModes:  intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: red
XLoad*highlight: black
XLoad*borderWidth: 0
emacs*Geometry:  80x65-0-0
emacs*Background:  rgb:5b/76/86
emacs*Foreground:  white
emacs*Cursor:  white
emacs*BorderColor:  white
emacs*Font:  6x10
xmag*geometry: -0-0
xmag*borderColor:  white
```

If these resources were stored in a file called *.Xresources* in your home directory, they could be added to any existing resources in the server with the following command:

```
% xrdb -merge $HOME/.Xresources
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of automatically loading resources. See Chapter 11, *Setting Resources*, for more information.

Examples

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command's reference page.

```
% xrdb -load $HOME/.Xresources
% xmodmap -e 'keysym BackSpace = Delete'
% mkfontdir /usr/local/lib/X11/otherfonts
% xset fp+ /usr/local/lib/X11/otherfonts
% xmodmap $HOME/.keymap.km
% xsetroot -solid 'rgbi:.8/.8/.8'
% xset b 100 400 c 50 s 1800 r on
```

```
% xset q
% twm
% xmag
% xclock -geometry 48x48-0+0 -bg blue -fg white
% xeyes -geometry 48x48-48+0
% xbiff -update 20
% xlsfonts '*helvetica*'
% xwininfo -root
% xdpyinfo -display joesworkstation:0
% xhost -joesworkstation
% xrefresh
% xwd | xwud
% bitmap companylogo.bm -size 32x32
% xcalc -bg blue -fg magenta
% xterm -geometry 80x66-0-0 -name myxterm
% xon filesysmachine xload
```

Diagnostics

A wide variety of error messages are generated from various programs. The default error handler in Xlib (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in */usr/lib/X11/XErrorDB*. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsics encounter errors converting resource strings to the appropriate internal format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (e.g., color versus monochrome, lots of fonts versus very few, etc.), although it can pose problems in trying to determine why an application might be failing. This behavior can be overridden by setting the `StringConversionWarning` resource.

To force the X Toolkit Intrinsics to always print string conversion error messages, the following resource should be placed at the top of the file that is loaded onto the RESOURCE_MANAGER property using the *xrdb* program (frequently called *.Xresources* or *.xrdb* in the user's home directory):

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

See Also

XConsortium(1), XStandards(1), Xsecurity(1), appres, auto_box(1), bdftopcf, beach_ball(1), bitmap, editres, fs, fsinfo, fslsfonts, fstobdf, ico(1), imake(1), listres, lndir(1), makedepend(1), maze(1), mkdirhier(1), mkfontdir, mwm, oclock, plbpex(1), puzzle(1), resize, showfont, showrgb, twm, viewres, x11perf(1), x11perfcomp(1), xauth, xbiff, xcalc, xclipboard, xclock, xcmsdb, xcmstest(1), xconsole, xcutsel, xditview, xdm, xdpr, xdpyinfo, xedit, xev, xeyes, xfd, xfontsel, xgas(1), xgc(1), xhost, xinit, xkill, xload, xlogo, xlsatoms, xlsclients, xlsfonts, xmag, xman, xmh, xmkmf(1), xmodmap, xon(1), xpr, xprop, xrdb, xrefresh, xset, xsetroot, xstdcmap, xterm, xwd, xwininfo, xwud, Xserver, Xdec(1), XmacII(1), Xmips(1), Xqdss(1), Xqvss(1), Xsun(1), X386(1), kbd_mode(1), Volume One, *Xlib Programming Manual*; Volume Two, *Xlib Reference Manual*; Volume Four, *X Toolkit Intrinsics Programming Manual*; Volume Five, *X Toolkit Intrinsics Reference Manual*; Volume Eight, *X Window System Administrator's Guide*.

Copyright

Trademarks

X Window System is a trademark of MIT.

Authors

A cast of thousands, literally. The MIT Release 5 distribution is brought to you by the MIT X Consortium. The names of all people who made it a reality will be found in the individual documents and source files. The staff members at MIT responsible for this release are: Donna Converse (MIT X Consortium), Stephen Gildea (MIT X Consortium), Susan Hardy (MIT X Consortium), Jay Hersh (MIT X Consortium), Keith Packard (MIT X Consortium), David Sternlicht (MIT X Consortium), Bob Scheifler (MIT X Consortium), and Ralph Swick (Digital/MIT Project Athena).

Name

X – X Window System server.

Syntax

X [:*displaynumber*] [*options*] [*ttyname*]

Description

X is the generic name for the X Window System server. It is frequently a link to or a copy of the appropriate server binary for driving the most frequently used server on a given machine.

Starting the Server

The server is usually started from the X Display Manager program, *xdm*. This utility is run from the system boot files and takes care of keeping the server running, prompting for user-names and passwords, and starting up user sessions. It is easily configured for sites that wish to provide consistent interfaces for novice users (loading convenient sets of resources, starting up a window manager, clock, and a wide selection of terminal emulator windows).

Installations that run more than one window system will still need to use the *xinit* utility. However, *xinit* is to be considered a tool for building startup scripts and is not intended for use by end users. Site administrators are *strongly* urged to use *xdm*, or to build more friendly interfaces for novice users.

When the X server starts up, it takes over the display. If you are running on a workstation whose console is the display, you cannot log into the console while the server is running.

Network Connections

The X server supports connections made using the following reliable byte-streams:

TCP/IP The server listens on port (6000+*n*), where *n* is the display number.

UNIX Domain

The X server uses */tmp/.X11-unix/X**n*** as the filename for the socket, where ***n*** is the display number.

DECnet

The server responds to connections to object *X$X**n***, where ***n*** is the display number.

Options

All of the X servers accept the following command-line options:

-a *number*

Sets pointer acceleration (i.e., the ratio of how much is reported to how much the user actually moved the pointer).

-auth *authorization_file*

Specifies a file that contains a collection of authorization records used to authenticate access. See also *xdm* and *xsecurity*.

`bc` Disables certain kinds of error checking, for bug compatibility with previous releases (e.g., to work around bugs in Release 2 and Release 3 versions of *xterm* and the toolkits). Use of this option is discouraged.

`-bs` Disables backing store support on all screens.

`-c` Turns off key-click.

`c` *volume*
Sets key-click volume (allowable range: 0-8).

`-cc` *class*
Sets the visual class for the root window of color screens. The class numbers are as specified in the X protocol. Not obeyed by all servers.

`-co` *filename*
Sets the name of the RGB color database.

`-dpi` *resolution*
Sets the resolution of the screen, in dots-per-inch. To be used when the server cannot determine the screen size from the hardware.

`-f` *volume*
Sets beep (bell) volume (allowable range: 0-7).

`-fc` *cursor_font*
Sets the default cursor font.

`-fn` *font*
Sets the default font.

`-fp` *font_path*
Sets the search path for fonts. This path is a comma-separated list of directories that the X server searches for font databases.

`-help` Prints a usage message.

`-I` Causes all remaining command-line arguments to be ignored.

`-ld` *kilobytes*
Sets the data space limit of the server to the specified number of kilobytes. A value of zero makes the stack size as large as possible. The default value of –1 leaves the stack space limit unchanged. (Not available in all operating systems.)

`-lf` *files*
Sets the number-of-open-files limit of the server to the specified number. A value of zero makes the limit as large as possible. The default value of –1 leaves limit unchanged. This option is not available in all operating systems. (Available as of Release 5.)

`-ls` *`kilobytes`*
Sets the stack space limit of the server to the specified number of kilobytes. A value of zero makes the stack size as large as possible. The default value of –1 leaves the stack space limit unchanged. (Not available in all operating systems.)

`-logo` Turns on the X Window System logo display in the screen saver. There is currently no way to change this from a client.

`nologo`
Turns off the X Window System logo display in the screen saver. There is currently no way to change this from a client.

`-p` *`minutes`*
Sets screen saver pattern cycle time, in minutes.

`-r` Turns off auto-repeat.

`r` Turns on auto-repeat.

`-s` *`minutes`*
Sets screen saver timeout, in minutes.

`-su` Disables save under support on all screens.

`-t` *`numbers`*
Sets pointer acceleration threshold, in pixels (i.e., after how many pixels pointer acceleration should take effect).

`-to` *`seconds`*
Sets default screen saver timeout, in seconds.

`v` Sets video-off screen saver preference.

`-v` Sets video-on screen saver preference.

`-wm` Forces the default backing-store of all windows to be `WhenMapped`; a cheap-trick way of getting backing-store to apply to all windows.

`-x` *`extension`*
Loads the specified extension at *init*. (Not supported in most implementations.)

XDMCP-specific Options

You can also have the X server connect to *xdm* using XDMCP. Although this is not typically useful, because it doesn't allow *xdm* to manage the server process, it can be used to debug XDMCP implementations, and servers as a sample implementation of the server side of XDMCP. For more information on this protocol, see the *X Display Manager Control Protocol* specification (available in the MIT source in doc/XDMCP/xdmcp.ms). The following options control the behavior of XDMCP:

`-query` *`host_name`*
Enables XDMCP and sends `Query` packets to the specified host.

`-broadcast`
Enables XDMCP and broadcasts `BroadcastQuery` packets to the network. The first responding display manager will be chosen for the session.

`-indirect` *`host_name`*
Enables XDMCP and sends `IndirectQuery` packets to the specified host.

`-port` *`port_num`*
Specifies an alternate port number for XDMCP packets. Must be specified before any `-query`, `-broadcast`, or `-indirect` options.

`-once` Makes the server exit after the first session is over. Normally, the server keeps starting sessions, one after the other.

`-class` *`display_class`*
XDMCP has an additional display qualifier used in resource lookup for display-specific options. This option sets that value; by default, it is "MIT-Unspecified" (not a very useful value).

`-cookie` *`xdm_auth_bits`*
When testing XDM-AUTHENTICATION-1, a private key is shared between the server and the manager. This option sets the value of that private data (not that it's very private, being on the command line).

`-displayID` *`display_ID`*
Yet another XDMCP-specific value, this one allows the display manager to identify each display so that it can locate the shared key.

Many servers also have device-specific command-line options. For more details, see the manual pages for the individual servers.

Security

The X server implements a simplistic authorization protocol, MIT-MAGIC-COOKIE-1, which uses data private to authorized clients and the server. This is a rather trivial scheme; if the client passes authorization data which is the same as the data the server has, it is allowed access. This scheme is worse than the host-based access control mechanisms in environments with unsecure networks because it allows any host to connect, given that it has discovered the private key. But in many environments, this level of security is better than the host-based scheme, because it allows access to be controlled per-user instead of per-host.

In addition, the server provides support for a DES-based authorization scheme, XDM-AUTHORIZATION-1, which is more secure (given a secure key distribution mechanism). This authorization scheme can be used in conjunction with XDMCP's authentication scheme, XDM-AUTHENTICATION-1, or in isolation.

The authorization data is passed to the server in a private file named with the `-auth` command-line option. Each time the server is about to accept the first connection after a reset (or when the server is starting), it reads this file. If this file contains any authorization records, the local host is not automatically allowed access to the server, and only clients which send one of

the authorization records contained in the file in the connection setup information will be allowed access. Maintenance of this file, and distribution of its contents to remote sites for use there, is left as an exercise for the reader.

The server also provides support for SUN-DES-1, using Sun's Secure RPC. It involves encrypting data with the X server's public key. See the *Xsecurity* reference page in the MIT source distribution.

The X server also uses a host-based access control list for deciding whether or not to accept connections from clients on a particular machine. If no other authorization mechanism is being used, this list initially consists of the host on which the server is running as well as any machines listed in the file */etc/X**n**.hosts*, where ***n*** is the display number of the server. Each line of the file should contain either an Internet hostname (e.g., expo.lcs.mit.edu) or a DECnet hostname in double colon format (e.g., hydra::). There should be no leading or trailing spaces on any lines. For example:

```
joesworkstation
corporate.company.com
star::
bigcpu::
```

Users can add or remove hosts from this list and enable or disable access control using the *xhost* command from the same machine as the server.

The X protocol intrinsically does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen. Sites that have authentication and authorization systems (such as Kerberos) might wish to make use of the hooks in the libraries and the server to provide additional security.

Signals

The X server attaches special meaning to the following signals:

SIGHUP — Causes the server to close all existing connections, free all resources, and restore all defaults. It is sent by the display manager whenever the main user's primary application (usually an *xterm* or window manager) exits to force the server to clean up and prepare for the next user.

SIGTERM — Causes the server to exit cleanly.

SIGUSR1 — This signal is used quite differently from either of the above. When the server starts, it checks to see if it has inherited SIGUSR1 as SIG_IGN instead of the usual SIG_DFL. In this case, the server sends a SIGUSR1 to its parent process, after it has set up the various connection schemes. *xdm* uses this feature to recognize when it is possible to connect to the server.

Fonts

Fonts are usually stored as individual files in directories. The X server can obtain fonts from directories and/or from font servers. The list of directories in which the server looks when trying to open a font is controlled by the *font path*. Although most sites will choose to have the X server start up with the appropriate font path (using the `-fp` option mentioned above), it can be overridden using the *xset* program.

The default font path for the X server contains four directories:

/usr/lib/X11/fonts/misc

This directory contains several miscellaneous bitmap fonts that are useful on all systems. It contains a family of fixed-width fonts, a family of fixed-width fonts from Dale Schumacher, several Kana fonts from Sony Corporation, two JIS Kanji fonts, two Hangul fonts from Daewoo Electronics, two Hebrew fonts from Joseph Friedman, the standard cursor font, two cursor fonts from Digital Equipment Corporation, and cursor and glyph fonts from Sun Microsystems. It also has font name aliases for the fonts, including *fixed* and *variable*.

/usr/lib/X11/fonts/Speedo

This directory contains outline fonts for Bitstream's Speedo rasterizer. A single font face, in normal, bold, italic, and bold italic, is provided, contributed by Bitstream, Inc.

/usr/lib/X11/fonts/75dpi

This directory contains bitmap fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc. for 75 dots per inch displays. An integrated selection of sizes, styles, and weights is provided for each family.

/usr/lib/X11/fonts/100dpi

This directory contains 100 dots per inch versions of some of the fonts in the *75dpi* directory.

Font databases are created by running the *mkfontdir* program in the directory containing the compiled versions of the fonts (the *.pcf* files). Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. If *mkfontdir* is not run, the server will not be able to find any fonts in the directory.

Diagnostics

Too numerous to list them all. If run from *init*(8), errors are typically logged in the file */usr/adm/Xnmsgs*.

Files

*/etc/X**n**.hosts* Initial access control list.

/usr/lib/X11/fonts/misc, /usr/lib/X11/fonts/75dpi, /usr/lib/X11/fonts/100dpi
Font directories.

/usr/lib/x11/fonts/Speedo
Outline font directory

/usr/lib/X11/fonts/PEX
PEX font directory.

/usr/lib/X11/rgb.txt Color database.

*/tmp/.X11-unix/X**n*** UNIX domain socket.

*/usr/adm/X**n**msgs* Error log file.

See Also

X, bdf to pcf, mkfontdir, fs, xauth, xdm, xhost, xinit, xset, xsetroot, xterm, ttys(5), init(8); Volume Zero, *X Protocol Reference Manual*; the following papers in the source distribution: *Definition of the Porting Layer for the X v11 X Server; Strategies for Porting the X v11 X Server; and Godzilla's Guide to Porting the X V11 X Server.*

Bugs

The option syntax is inconsistent with itself and *xset*.

The acceleration option should take a numerator and a denominator like the protocol.

If *X* dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The color database is missing a large number of colors.

Authors

The X server was originally written by Susan Angebranndt, Raymond Drewry, Philip Karlton, and Todd Newman, of Digital Equipment Corporation, with support from a large cast. It has since been extensively rewritten by Keith Packard and Bob Scheifler of MIT.

appres

List Application Resources

Name

appres – list application resource database.

Syntax

appres [[*class_name*][*instance_name*]] [*options*]

Description

The *appres* client prints the resources seen by an application (or a subhierarchy of an application) with the specified *class_name* and *instance_name*. It is used to help determine which resources a particular program would load from the various sources of resource specifications.

Note that *appres* doesn't really know anything about classes and instances as they may be defined by the client itself. As a result, it takes no account of conflicts between different resource settings, or their correctness. It simply loads the resource database into a temporary file and does a string comparison on the strings:

```
[*.]
class_name[*.]
instance_name[*.]
```

(where [*.] means either * or .) and then prints out the lines that match. Basically, *appres* searches for occurrences of any class and/or instance name supplied to it. In addition, *appres* searches for resources not assigned to a particular client—i.e., resources beginning with an asterisk or a dot. (These resources may or may not apply to the client whose class and instance names you supply.)

For example:

```
% appres XTerm
```

would list the resources that include the classname XTerm, as well as any resources beginning with an asterisk or dot.

To also match a particular instance name, you can enter both a class and an instance name, as in the following:

```
% appres XTerm myxterm
```

In this case, *appres* would list the resources that include any of the following terms: the classname XTerm; the instance name myxterm; or an initial asterisk or dot.

As of Release 5, *appres* recognizes the X Toolkit option -name. Thus, the alternative syntax:

```
% appres XTerm -name myxterm
```

is also acceptable and will produce the same output as the preceding example.

If no application class is specified, the class -AppResTest- (which has no defaults) is used. (Prior to Release 5, this dummy class name was -NoSuchClass-.)

Keep in mind the limitations of supplying only one argument (either a class name or an instance name). For example, if resources are specified in the database for the instance name `xterm`, typing `appres XTerm` will not list them; and if they are specified for class name `XTerm`, typing `appres xterm` will not list them. To be safe, you should specify both the class name and the instance name.

As of Release 5, *appres* also accepts hierarchical class and instance names as input. Hypothetically, this allows you to list the resources that match a particular sublevel of an application's widget hierarchy. (This could be very useful with a complex application.)

To list the resources that may apply to part of the widget hierarchy, you provide *appres* with both a hierarchical class and an instance name. The number of class and instance components must be equal. (Note that the instance name should not be specified with the toolkit `-name` option.) For example, the command:

```
% appres Xman.TopLevelShell.Form xman.topBox.form
```

will list the resources that *may* apply to widgets within *xman*'s `topBox` hierarchy.

Note that in attempting to match hierarchical class and instance names, *appres* suffers from the same limitations it does when attempting to match single class and/or instance names. The *appres* client simply compares text strings; it does not distinguish valid from invalid resources. If you supply multiple components, *appres* returns any resource that includes any one of the components; or any resource not assigned to a particular client (i.e., any resource beginning with an asterisk or a dot.)

You can limit the matching to resources that apply to a specific widget in the hierarchy by using the `-1` option. For example, the command:

```
% appres XTerm.VT100 xterm.vt100 -1
```

will list the resources that may match the *xterm* `vt100` widget.

In practice, use of the `-1` option limits the matching to resource names having the same number of components, or fewer, as the names the user specifies. In the preceding example, the matching is limited to resources of two components or fewer. (This practice has nothing to do with the digit 1 in the `-1` option; this number is a literal, not a variable.) Note, however, that limiting the resources that can be matched does not eliminate the problem of *appres* returning inapplicable resources.

For more information on the use of *appres*, see Chapter 11, *Setting Resources*.

Options

Note that options should follow the *`class_name`* (and *`instance_name`* if any). *appres* accepts the following application-specific option:

`-1` Lists only the resources matching a specific level in the widget hierarchy (the level given on the command line). (Available as of Release 5.)

As of Release 5, *appres* also *recognizes* all of the standard X Toolkit options (i.e., the program will run); however, since *appres* is not a window-based application, it *uses* only the following options:

`-name` *app_name*
: Specifies the instance name under which resources for the application should be found. (Available as of Release 5.)

`-xrm` *resource*
: Specifies that, in addition to the current application resources, *appres* should return the *resource* specified as an argument to `-xrm`, if that resource would apply to the *class_name* or *instance_name*. You must specify both a *class_name* and an *instance_name* in order to use the `-xrm` option. (Note that `-xrm` does not actually load any resources.)

Without any arguments, *appres* returns those resources that might apply to any application (for example, those beginning with an asterisk in your *.Xresources* file).

See Also

X, xrdb, editres, listres; Chapter 11, *Setting Resources*.

Author

Jim Fulton, MIT X Consortium.

bdftopcf

Name

bdftopcf – convert font from Bitmap Distribution Format to Portable Compiled Format.

Syntax

bdftopcf [*options*] *fontfile*.**bdf**

Description

bdftopcf is the Release 5 font compiler. It converts Bitmap Display Fonts (BDF) to Portable Compiled Format (PCF), which can be read by any architecture. Note, however, that the PCF file is structured to allow one particular architecture to read them directly without reformatting. This allows fast reading on the appropriate machine, but the files are still portable (albeit read more slowly) on other machines. See Volume Eight, *X Window System Administrator's Guide*, for more information.

Options

`-i` Inhibits the normal computation of ink metrics. When a font has glyph images which do not fill the bitmap image (i.e., the "on" pixels don't extend to the edges of the metrics) *bdftopcf* computes the actual ink metrics and places them in the *.pcf* file; the `-t` option inhibits this behavior.

`-l` Sets the font bit order to LSB (least significant bit) first. The left most bit on the screen will be in the lowest valued bit in each unit.

`-L` Sets the font byte order to LSB first. All multi-byte data in the file (metrics, bitmaps and everything else) will be written least significant byte first.

`-m` Sets the font bit order to MSB (most significant bit) first. Bits for each glyph will be placed in this order; i.e., the left most bit on the screen will be in the highest valued bit in each unit.

`-M` Sets the font byte order to MSB first. All multi-byte data in the file (metrics, bitmaps, and everything else) will be written most significant byte first.

`-o` *output_filename*
By default *bdftopcf* writes the pcf file to standard output; this option gives the name of a file to be used instead.

`-p`*n* Sets the font glyph padding. Each glyph in the font will have each scanline padded in to a multiple of *n* bytes, where *n* is `1`, `2`, `4`, or `8`.

`-t` When this option is specified, *bdftopcf* will convert fonts into "terminal" fonts when possible. A terminal font has each glyph image padded to the same size; the X server can usually render these types of fonts more quickly.

`-u`*n* Sets the font scanline unit. When the font bit order is different from the font byte order, the scanline unit *n* describes what unit of data (in bytes) are to be swapped; the unit *n* can be `1`, `2` or `4` bytes.

See Also

X; Chapter 6, *Font Specification*; Volume Eight, *X Window System Administrator's Guide*; Appendix M, *Logical Font Description Conventions*, in Volume Zero, *X Protocol Reference Manual*. Also see the document *Bitmap Distribution Format*, in the MIT distribution.

Author

Keith Packard, MIT X Consortium.

Name

bdftosnf – BDF to SNF font compiler for X11.

Syntax

bdftosnf [*options*] *bdf_file*

Description

As of Release 5, this program is no longer supported as part of the standard distribution of X. Use *bdftopcf* instead. The *bdftosnf* reference page is included merely for continuity.

bdftosnf reads a Bitmap Distribution Format (BDF) font from the specified file (or from standard input if no file is specified) and writes an X11 Server Natural Format (SNF) font to standard output. See Volume Eight, *X Window System Administrator's Guide*, for more information.

Options

bdftosnf accepts the following options:

-p*number*
: Forces the glyph padding to a specific *number*. The legal values are 1, 2, 4, and 8.

-u*number*
: Forces the scanline unit padding to a specific *number*. The legal values are 1, 2, and 4.

-m
: Forces the bit order to most significant bit first.

-l
: Forces the bit order to least significant bit first.

-M
: Forces the byte order to most significant byte first.

-L
: Forces the byte order to least significant byte first.

-w
: Prints warnings if the character bitmaps have bits set to one outside of their defined widths.

-W
: Prints warnings for characters with an encoding of –1; the default is to silently ignore such characters.

-t
: Expands glyphs in "terminal-emulator" fonts to fill the bounding box.

-i
: Suppresses computation of correct ink metrics for "terminal-emulator" fonts.

See Also

X, Xserver; Volume Eight, *X Window System Administrator's Guide*; Appendix M, *Logical Font Description Conventions*, in Volume Zero, *X Protocol Reference Manual*. Also see the document *Bitmap Distribution Format*, in the MIT distribution.

bitmap

Name

bitmap, bmtoa, atobm – bitmap editor and conversion utilities.

Syntax

bitmap [*options*] [*filename*] [*basename*]

bmtoa [*options*] [*filename*]

atobm [*options*] [*filename*]

Description

bitmap allows you to create and edit small bitmaps that you can use as backgrounds, clipping regions, tile and stipple patterns, icons, and pointers. A bitmap is a grid of pixels, or picture elements, each of which is white, black, or, in the case of color displays, a color. See Chapter 7, *Graphics Utilities*, for instructions on using *bitmap*.

The *bmtoa* and *atobm* filters convert *bitmap* files to and from ASCII strings. They are most commonly used to print out bitmaps quickly and to generate versions for inclusion in text. Chapter 7 describes how to convert a font character to a bitmap using *atobm*.

The window that *bitmap* creates has three sections (see Figure 7-1 in Part One of this guide). The checkerboard grid is a magnified version of the bitmap you are editing. Each square represents a single bit in the picture being edited. Squares on the grid can be set, cleared, or inverted directly with the buttons on the pointer. Command buttons to perform higher-level operations, such as drawing lines and circles, are provided to the left of the grid. You can invoke these command buttons by clicking on them with the first pointer button. Across the top of the window is a menu bar that provides a File menu and an Edit menu.

You can display an actual size representation of the bitmap image (as it would appear both normally and inverted) by pressing the Meta-I key combination. You are free to move the image popup out of the way to continue editing. Clicking the first pointer button in the popup window or typing Meta-I again will remove the actual size bitmap image.

If the bitmap is to be used for defining a cursor, one of the squares in the image may be designated as the *hot spot*. This determines where the cursor is actually pointing. For cursors with sharp tips (such as arrows or fingers), this is usually at the end of the tip; for symmetric cursors (such as crosses or bullseyes), this is usually at the center.

Bitmaps are stored as small C code fragments suitable for including in applications. They provide an array of bits as well as symbolic constants giving the width, height, and hot spot (if specified) that may be used in creating cursors, icons, and tiles. A selection of commonly-used bitmaps is generally stored in */usr/include/X11/bitmaps* on UNIX systems.

You give the size in pixels of the bitmap to be created (and consequently, the number of cells in the bitmap editing area) using the `-size` option. Existing bitmaps should be edited at their current size. The default size for new bitmaps is 16 × 16. (This is a little small—an icon such as the mailbox displayed by *xbiff* is 48 × 48 pixels.) See Chapter 7 for a discussion of *bitmap* window and image size issues.

When you run *bitmap* without a filename or with a new filename, the window will display a blank image area, suitable for you to begin editing. To edit a bitmap image, you can use one of the editing command buttons or use the pointer commands to change individual grid squares. (Chapter 7 describes both ways of editing; also see "Command Buttons for Drawing" and "Changing Grid Squares Using the Pointer" later in this reference page.)

Options: bitmap

bitmap accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. In addition, *bitmap* accepts the following application-specific options:

`-axes, +axes`
: Turns the major axes on or off.

`-dashed, +dashed`
: Turns dashing for the frame and grid lines on or off.

`-dashes` *`filename`*
: Specifies the bitmap to be used as a stipple for dashing.

`-fr` *`color`*
: Specifies the color used for the frame and grid lines.

`-grid, +grid`
: Turns the grid lines on or off.

`-gt` *`pixels`*
: Grid tolerance. If the square dimensions fall below the specified value, grid will be automatically turned off. The default is 8 (pixels square).

`-hl` *`color`*
: Specifies the color to be used for highlighting.

`-proportional, +proportional`
: Turns proportional mode on or off. If proportional mode is on, square width is equal to square height. If proportional mode is off, *bitmap* will use the smaller square dimension, if they were initially different.

`-sh` *`pixels`*
: Specifies the height of squares in pixels.

`-size` *`WIDTHxHEIGHT`*
: Specifies size of the grid in squares. *`WIDTHxHEIGHT`* are two numbers, separated by the letter "x", which specify the dimensions of the checkerboard grid within the *bitmap* window (e.g., `9x13`). The first number is the grid's width; the second number is its height. The default is `16x16`.

`-stipple` *`filename`*
: Specifies the bitmap to be used as a stipple for highlighting.

`-stippled, +stippled`
: Turns stippling of highlighted squares on or off.

`-sw` *`pixels`*
Specifies the width of squares in pixels.

The *bitmap* editor also accepts the following arguments:

`basename`
Specifies the basename to be used in the C code output file. If it is different than the basename in the working file, *bitmap* will change the name of the file when saving.

`filename`
Specifies the bitmap to be initially loaded into the program. If the file does not exist, *bitmap* will assume it is a new file.

Options: bmtoa

The *bmtoa* (bitmap to array) conversion program accepts the following option:

`-chars` *`cc`*
Specifies the pair of characters to use in the string version of the bitmap. The first character is used for 0 bits and the second character is used for 1 bits. The default is to use dashes (–) for 0s and number signs (#) for 1s.

Options: atobm

The *atobm* (array to bitmap) conversion program accepts the following options:

`-chars` *`cc`*
Specifies the pair of characters to use when converting string bitmaps into arrays of numbers. The first character represents a 0 bit and the second character represents a 1 bit. The default is to use dashes (–) for 0s and number signs (#) for 1s.

`-name` *`variable`*
Specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *`filename`* command-line argument or leave it blank if the standard input is read.

`-xhot` *`number`*
Specifies the X coordinate of the hot spot. Only positive values are allowed. By default, no hot spot information is included.

`-yhot` *`number`*
Specifies the Y coordinate of the hot spot. Only positive values are allowed. By default, no hot spot information is included.

Resources

For a table of settable resources, see the section "Bitmap Widget Resources" later in this reference page. Appendix G, *Widget Resources*, describes the resources that can be set for other widgets in the application. (See "Widget Hierarchy" later in this reference page.)

Changing Grid Squares Using the Pointer

You can set, clear, or invert grid squares by pointing to them and clicking or dragging using one of the buttons as indicated below. Setting a grid square corresponds to setting a bit in the bitmap image to 1. Clearing a grid square corresponds to setting a bit in the bitmap image to 0. Inverting a grid square corresponds to changing a bit in the bitmap image from 0 to 1 or 1 to 0, depending upon its previous state. You can change multiple squares at once by holding the button down and dragging the cursor across them. The default behavior of pointer buttons is:

Button 1 (usually the left)
: Sets one or more grid squares to the foreground color and sets the corresponding bits in the bitmap to 1.

Button 2 (usually the middle)
: Inverts one or more grid squares. The corresponding bit or bits in the bitmap are inverted (1s become 0s and 0s become 1s).

Button 3 (usually the right)
: Clears one or more grid squares (sets them to the background color) and sets the corresponding bits in the bitmap to 0.

For pointers with additional buttons, the fourth and fifth also clear the grid square(s).

This default behavior can be changed by setting the button function resources:

```
bitmap*button1Function:  Set
bitmap*button2Function:  Clear
bitmap*button3Function:  Invert
```

Note that the button function applies to all drawing commands, including copying, moving and pasting, flood filling and setting the hot spot.

Command Buttons for Drawing

bitmap provides 27 command buttons to assist you in drawing. The buttons are located to the left of the editing grid and their functions are summarized below. See Chapter 7 for more complete instructions. Note that most of the actions can also be performed using keyboard shortcuts (accelerators) while the pointer rests inside the editing grid or the surrounding whitespace.

Clear
: Clears all bits in the bitmap image. Sets all of the grid squares to the background color. Typing c while the pointer rests inside the grid or the surrounding whitespace has the same effect.

Set
: Sets all bits in the bitmap image. Sets all of the grid squares to the foreground color. Typing s has the same effect.

Invert
: Inverts all bits in the bitmap image. The grid squares will be inverted appropriately. Typing i has the same effect.

Mark
: Allows you to mark an area of the grid by dragging out a rectangular shape in the highlighting color. Once the area is marked, you can perform a number of other

operations on it. (See Up, Down, Left, Right, Rotate, Flip, Cut, etc.) Only one marked area can be present at any time. If you attempt to mark another area, the old mark will vanish. The same effect can be achieved by pressing the Shift key and the first pointer button simultaneously and then dragging to highlight a rectangle in the grid window. Press Shift and click the second pointer button to mark the entire grid area.

Unmark Causes the marked area to vanish. You can perform the same action by pressing Shift and clicking the third pointer button.

Copy Allows you to copy an area of the grid from one location to another. If there is no marked grid area displayed, Copy first behaves just like Mark. Once there is a marked grid area displayed in the highlighting color, Copy has two alternative behaviors. If you press a pointer button inside the marked area, you can drag a rectangle representing the marked area to the desired location. When you release the pointer button, the area is copied. If you click outside the marked area, Copy will assume that you wish to mark a different region of the bitmap image and it will behave like Mark again.

Move Allows you to move an area of the grid from one location to another. Its behavior resembles the behavior of Copy command, except that the marked area will be moved instead of copied.

Flip Horizontally
Flips the bitmap image with respect to the horizontal axes. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing h has the same effect.

Up Moves the bitmap image one pixel up. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing the up arrow key has the same effect.

Flip Vertically
Flips the bitmap image with respect to the vertical axes. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing v has the same effect.

Left Moves the bitmap image one pixel to the left. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing the left arrow key has the same effect.

Fold Folds the bitmap image so that the opposite corners become adjacent. This is useful when creating bitmap images for tiling. Pressing f has the same effect.

Right Moves the bitmap image one pixel to the right. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing the right arrow key has the same effect.

Rotate Left

Rotates the bitmap image 90 degrees to the left (counterclockwise.) If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing l has the same effect.

Down Moves the bitmap image one pixel down. If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing the down arrow has the same effect.

Rotate Right

Rotates the bitmap image 90 degrees to the right (clockwise). If a marked area of the grid is highlighted, it will operate only inside the marked area. Pressing r has the same effect.

Point Changes the grid squares underneath the pointer according to the guidelines explained in "Changing Grid Squares Using the Pointer." If you press a button and drag the pointer, the line may not be continuous depending on the speed of your system and frequency of pointer motion events.

Curve Changes the grid squares underneath the pointer according to the guidelines explained in "Changing Grid Squares Using the Pointer." The Curve command ensures that when you press a pointer button and drag, the line will be continuous. However,if your system is slow or *bitmap* receives very few pointer motion events, it might behave erratically.

Line Allows you to draw the straightest possible line between two grid squares. Once you press a pointer button in the grid window, *bitmap* will highlight the line from the square where the pointer button was initially pressed to the square where the pointer is located. When you release the pointer button, the highlighting will disappear and the actual line will be drawn.

Rectangle

Allows you to draw a rectangle. Once you press a pointer button in the grid window, *bitmap* will highlight the rectangle from the square where the pointer button was initially pressed to the square where the pointer is located. When you release the pointer button, the highlighting will disappear and the actual rectangle will be drawn.

Filled Rectangle

Performs the same function as Rectangle, except that the rectangle is filled (rather than outlined).

Circle Allows you to draw a circle. When you press a pointer button in the grid, that square becomes the center of the circle. You continue to hold the pointer button down and drag the pointer away from the center point to indicate a point on the circumference. An outline follows the pointer. When you release the pointer button, the highlighting will disappear and the actual circle will be drawn.

Filled Circle
: Performs the same function as Circle, except that the circle is filled (rather than outlined).

Flood Fill
: Fills any closed shape you click on. If a shape is open, you will "flood" a larger area than you intend. Diagonally adjacent squares are not considered to form a single shape.

Set Hot Spot
: Designates one square in the grid as the hot spot if this bitmap image is to be used for defining a cursor. Pressing a pointer button in the desired square causes a diamond shape to be displayed.

Clear Hot Spot
: Removes any designated hot spot from the bitmap image.

Undo
: Undoes the last executed command. You can only recover the last action performed; thus, pressing Undo after Undo will toggle the last action on and off.

File Menu

You can access the File menu commands by pressing the File button and selecting the appropriate menu entry or by pressing the Control key with another key. These commands deal with files and global bitmap parameters, such as size, basename, filename, etc. See Chapter 7 for more information.

New
: Clears the window so you can create a new image; prompts for a name for the new file. If you haven't saved the current file, the changes will be lost.

Load
: Dynamically loads another bitmap file into the editing window; if you haven't saved the current file, prompts you as to whether to save before loading the next file. The *bitmap* editor can edit only one file at a time. If you need interactive editing, run a number of editors and use cut and paste mechanism as described below.

Insert
: Inserts a bitmap file into the image currently being edited. After being prompted for the filename, click inside the grid window and drag the outlined rectangle to the location where you want to insert the new file.

Save
: Saves the bitmap image. It will not prompt for the filename unless it is said to be <none>. If you leave the filename undesignated or –, the output will be piped to standard output.

Save As
: Saves the bitmap image after prompting for a new filename. Use this menu item if you want to change the filename.

Resize
: Changes the dimensions of the editing grid to match dimensions you supply (*`widthxheight`*), without changing the size of the image. Thus, specifying a larger grid gives you more room to edit. Specifying a smaller grid may cause part of the current image to be truncated.

Rescale Changes the dimensions of the grid to match dimensions you supply (*widthx-height*) and changes the image so that the proportions (the ratio of the image to the grid) remain the same. Thus, if you specify a grid twice the size of the current one, both the grid and the image will be doubled. Rescale will not do antialiasing and specifying a smaller grid may cause part of the current image to be truncated. Feel free to add your own algorithms for better rescaling.

Filename
Lets you change the filename of the current file without changing the basename or saving the file. If you specify – for a filename, the output will be piped to standard output.

Basename
Lets you change the basename of the current file if you want one different from the filename.

Quit Terminates the *bitmap* application. If changes have been made and not saved, a dialog box will ask whether to save before quitting. This command is preferable to killing the process.

Edit Menu

The Edit menu commands can be accessed by pressing the Edit button and selecting the appropriate menu entry, or by pressing Meta key with another key. These commands deal with editing facilities such as the grid, axes, zooming, cut and paste, etc.

Image Displays a window showing what the bitmap being edited looks like at its actual size (both as it appears and in reverse video). You can move the window away to continue editing. Clicking the first pointer button on this window pops it down.

Grid Controls the grid in the editing area. If the grid spacing is below the value specified by the `gridTolerance` resource (8 by default), the grid will be automatically turned off. You can turn the grid on by selecting this menu item.

Dashed Controls the stipple for drawing the grid lines. Select this menu item to toggle the stipple (specified by the `dashes` resource or the `-dashes` option).

Axes Toggles diagonal axes. The axes simply assist in drawing; they are not part of the image. Off by default.

Stippled Toggles a stipple pattern to be used for highlighting within the editing area. The stipple specified by the `stipple` resource can be turned on or off by activating this command.

Proportional
Toggles proportional mode which forces proportional grid squares, regardless of the dimensions of the *bitmap* window. On by default.

Zoom Toggles zoom mode, which focuses in on a marked area of the image. (You can mark before or after selecting Zoom.) You can use all the editing commands and other utilities in the zoom mode. When you zoom out, Undo will undo the whole zoom session.

Cut — Cuts the contents of any marked area into the internal (application local) cut and paste buffer. The marked area is deleted from the current image, but is available to be pasted from the buffer. (If this was the last area marked, it is also available to be pasted into other applications via a global buffer.)

Copy — Copies the contents of any marked area into the internal (application local) cut and paste buffer. The marked area remains a part of the current image and is also available to be pasted from the buffer. (If this was the last area marked, it is also available to be pasted into other applications via a global buffer.)

Paste — Pastes the contents of the global buffer (the marked area in any *bitmap* or *xmag* application); if the global buffer is empty, this item pastes a copy of the contents of the internal cut and paste buffer. To place the copied image, press and hold the first pointer button in the editing area, drag the outlined image to the position you want, and then release the button.

Cut and Paste

bitmap supports two cut and paste mechanisms: an internal cut and paste buffer and the global X selection cut and paste buffer. The internal cut and paste is used when executing Copy and Move drawing commands and also Cut and Copy commands from the Edit menu. The global X selection cut and paste is used whenever there is a highlighted area of a bitmap image displayed anywhere on the screen. To copy a part of image from another bitmap editor, simply highlight the desired area by using the Mark command or pressing the Shift key and dragging the area with the first pointer button. When the selected area becomes highlighted, any other applications (such as *xterm*, etc.) that use the PRIMARY selection will discard their selection values and unhighlight the appropriate information. Now, use the Paste command from the Edit menu or press Control and any pointer button to copy the selected part of image into another (or the same) bitmap application. If you attempt to do this without a visible highlighted image area, the bitmap will fall back to the internal cut and paste buffer and paste whatever is stored there at the moment.

Widget Hierarchy

Below is the widget structure of the *bitmap* application. Indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name. All widgets except the bitmap widget are from the standard Athena widget set. See Appendix G, *Widget Resources*, for a list of resources that can be set.

```
Bitmap bitmap
    TransientShell image
        Box box
            Label normalImage
            Label invertedImage
    TransientShell input
        Dialog dialog
            Command okay
            Command cancel
```

```
TransientShell error
    Dialog dialog
        Command abort
        Command retry
TransientShell qsave
    Dialog dialog
        Command yes
        Command no
        Command cancel
Paned parent
    Form formy
        MenuButton fileButton
        SimpleMenu fileMenu
            SmeBSB  new
            SmeBSB  load
            SmeBSB  insert
            SmeBSB  save
            SmeBSB  saveAs
            SmeBSB  resize
            SmeBSB  rescale
            SmeBSB  filename
            SmeBSB  basename
            SmeLine line
            SmeBSB  quit
        MenuButton editButton
        SimpleMenu editMenu
            SmeBSB  image
            SmeBSB  grid
            SmeBSB  dashed
            SmeBSB  axes
            SmeBSB  stippled
            SmeBSB  proportional
            SmeBSB  zoom
            SmeLine line
            SmeBSB  cut
            SmeBSB  copy
            SmeBSB  paste
        Label status
    Pane pane
        Bitmap bitmap
        Form form
            Command clear
            Command set
            Command invert
            Toggle  mark
            Command unmark
            Toggle  copy
            Toggle  move
```

```
Command flipHoriz
Command up
Command flipVert
Command left
Command fold
Command right
Command rotateLeft
Command down
Command rotateRight
Toggle  point
Toggle  curve
Toggle  line
Toggle  rectangle
Toggle  filledRectangle
Toggle  circle
Toggle  filledCircle
Toggle  floodFill
Toggle  setHotSpot
Command clearHotSpot
Command undo
```

Bitmap Widget Resources

The Bitmap widget is a stand-alone widget for editing raster images. It is not designed to edit large images, although it may be used in that purpose as well. It can be freely incorporated with other applications and used as a standard editing tool.

Header file	`Bitmap.h`
Class	`bitmapWidgetClass`
Class Name	`Bitmap`
Superclass	`Bitmap`

The following are the resources provided by the Bitmap widget. All the Simple widget resources plus:

Name	Class	Type	Default Value
`foreground`	`Foreground`	`Pixel`	`XtDefaultForeground`
`highlight`	`Highlight`	`Pixel`	`XtDefaultForeground`
`framing`	`Framing`	`Pixel`	`XtDefaultForeground`
`gridTolerance`	`GridTolerance`	`Dimension`	`8`
`size`	`Size`	`String`	`32x32`
`dashed`	`Dashed`	`Boolean`	`True`
`dashes`	`Dashes`	`Bitmap`	`unspecified`
`grid`	`Grid`	`Boolean`	`True`
`stipple`	`Stipple`	`Bitmap`	`unspecified`
`stippled`	`Stippled`	`Boolean`	`True`
`proportional`	`Proportional`	`Boolean`	`True`

Name	Class	Type	Default Value
axes	Axes	Boolean	False
squareWidth	SquareWidth	Dimension	16
squareHeight	SquareHeight	Dimension	16
margin	Margin	Dimension	16
xHot	XHot	Position	NotSet (-1)
yHot	YHot	Position	NotSet (-1)
button1Function	Button1Function	DrawingFunction	Set
button2Function	Button2Function	DrawingFunction	Invert
button3Function	Button3Function	DrawingFunction	Clear
button4Function	Button4Function	DrawingFunction	Invert
button5Function	Button5Function	DrawingFunction	Invert
filename	Filename	String	none
basename	Basename	String	none

Color

If you would like bitmap to be viewable in color, include the following in the #ifdef COLOR section of the file you read with xrdb:

```
*customization:    -color
```

This line will cause bitmap to pick up the colors in the app-defaults color customization file, */usr/lib/X11/app-defaults/Bitmap-color*. See Chapter 11, *Setting Resources*, for more information.

Files

/usr/lib/X11/app-defaults/Bitmap
: Specifies required resources.

/usr/lib/X11/app-defaults/Bitmap-color
: Color customization file.

/usr/include/X11/bitmaps
: On many systems, standard bitmaps can be found in this directory.

Bugs

bitmap should really be implemented with a scrollable editing area, so that the size of the application can be completely independent of the size of the bitmap being edited.

If you move the pointer too fast while holding a pointer button down, some squares may be missed. This is caused by limitations in how frequently the X server can sample the pointer location.

See Also

xmag; Chapter 7, *Graphics Utilities*.

Author

Release 5 *bitmap* by Davor Matic, MIT X Consortium; Previous releases of *bitmap* by Ron Newman, MIT Project Athena; *bmtoa* and *atobm* by Jim Fulton, MIT X Consortium.

Name

editres – a dynamic resource editor for X Toolkit applications.

Synopsis

`editres` [*options*]

Description

editres is a tool that allows you to view the full widget hierarchy of any X Toolkit client that speaks the *editres* protocol. In addition, *editres* will help you construct resource specifications and allow you to apply the resources to the application and view the results dynamically. Once you're happy with a resource specification, you can request that *editres* append the specification to a resource file. See Chapter 11, *Setting Resources*, for instructions on using *editres*.

Using editres

editres provides a window consisting of the following four areas:

Menu Bar — A set of popup menus that allows you full access to the program's features.

Panner — The panner allows a more intuitive way to scroll the application tree display.

Message Area — Displays information to the user about the action that *editres* expects of her.

Application Widget Tree
: This area will be used to display the selected client's widget tree.

To begin an *editres* session, select the Get Widget Tree menu item from the command menu. This will change the pointer cursor to cross hair. You should now select the application you wish to look at by clicking on any of its windows. If this application understands the *editres* protocol, then *editres* will display the client's widget tree in its tree window. If the application does not understand the *editres* protocol, *editres* will inform you of this fact in the message area after a few seconds delay.

Once you have a widget tree you may now select any of the other menu options. The effect of each of these is described below.

Commands Menu

Get Widget Tree
: Allows the user to click on any client that speaks the *editres* protocol, and receive its widget tree.

Refresh Current Widget Tree
: *editres* only knows about the widgets that exist at the present time. Many applications create and destroy widgets "on-the-fly." Selecting this menu item will cause *editres* to ask the application to resend its widget tree, thus updating its information to the new state of the application.

  For example, *xman* only creates the widgets for its *topbox* when it starts up. None of the widgets for the manual page window are created until the user actually clicks on the Manual Page button. If you retrieved *xman*'s widget tree before the manual page

is active, you may wish to refresh the widget tree after the manual page has been displayed. This will also allow you to edit the manual page's resources.

Dump Widget Tree to a File
: For documenting applications, it is often useful to be able to dump the entire application widget tree to an ASCII file. This file can then be included in the manual page. When this menu item is selected, a popup dialog is activated. Type the name of the file in this dialog, and either select okay, or press Return. *editres* will now dump the widget tree to this file. To cancel the file dialog just select the cancel button.

Show Resource Box
: This command will popup a resource box for the current client. This resource box (described in detail below) will allow the user to see exactly which resources can be set for the widget that is currently selected in the widget tree display. Only one widget may be currently selected, if greater or fewer are selected *editres* will refuse to pop up the resource box, and put an error message in the Message Area.

Set Resource
: This command will popup a simple dialog box for setting an arbitrary resource on all selected widgets. You must type in the resource name, as well as the value. You can use the Tab key to switch between the resource name field and the resource value field.

Quit
: Exits *editres*.

Tree Menu Commands

The Tree menu contains several commands that allow operations to be performed on the widget tree.

Select Widget in Client
: This menu item allows you to select any widget in the application; *editres* will then highlight the corresponding element in the widget tree display. Once this menu item is selected, the pointer cursor will again turn to a crosshair, and you must click any pointer button in the widget you wish to have displayed. Since some widgets are fully obscured by their children, it is not possible to get to every widget this way, but this mechanism does give very useful feedback between the elements in the widget tree and those in the actual client.

Select All
Unselect All
Invert All
: These functions allow the user to select, unselect, or invert all widgets in the widget tree.

Select Children
Select Parents
: These functions select the immediate parent or children of each of the currently selected widgets.

Select Descendants
Select Ancestors

These functions select all parents or children of each of the currently selected widgets. This is a recursive search.

Show Widget Names
Show Class Names
Show Widget IDs
Show Widget Windows

When the tree widget is initially displayed, the labels of each widget in the tree correspond to the widget names. These functions will cause the label of *all* widgets in the tree to be changed to show the class name, ID, or window associated with each widget in the application. The widget IDs, and windows are shown as hex numbers.

In addition, there are keyboard accelerators for each of the Tree operations. If the input focus is over an individual widget in the tree, then that operation will only effect that widget. If the input focus is in the Tree background it will have exactly the same effect as the corresponding menu item.

The translation entries shown may be applied to any widget in the application. If that widget is a child of the Tree widget, then it will only affect that widget, otherwise it will have the same effect as the commands in the Tree menu.

Flash Active Widgets

This command is the inverse of the Select Widget in Client command, it will show the user each widget that is currently selected in the widget tree by flashing the corresponding widget in the application `numFlashes` three (by default) times in the `flashColor`.

Key	Option	Translation Entry
space	Unselect	Select(nothing)
w	Select	Select(widget)
s	Select	Select(all)
i	Invert	Select(invert)
c	Select Children	Select(children)
d	Select Descendants	Select(descendants)
p	Select Parent	Select(parent)
a	Select Ancestors	Select(ancestors)
N	Show Widget Names	Relabel(name)
C	Show Class Names	Relabel(class)
I	Show Widget IDs	Relabel(id)
W	Show Widget Windows	Relabel(window)
T	Toggle Widget/Class Name	Relabel(toggle)

Clicking button 1 on a widget adds it to the set of selected widgets. Clicking button 2 on a widget deselects all other widgets and then selects just that widget. Clicking button 3 on a widget toggles its label between the widget's instance name the widget's class name.

Using the Resource Box

The resource box contains five different areas. Each of the areas, as they appear on the screen from top to bottom, will be discussed.

The Resource Line

This area at the top of the resource box shows the current resource name exactly as it would appear if you were to save it to a file or apply it.

The Widget Names and Classes

This area allows you to select exactly which widgets this resource will apply to. The area contains four lines, the first contains the name of the selected widget and all its ancestors, and the more restrictive dot (.) separator. The second line contains less specific Class names of each widget, as well as the less restrictive star (*) separator. The third line contains a set of special buttons called Any Widget, which will generalize this level to match any widget. The last line contains a set of special buttons called Any Widget Chain, which will turn the single level into something that matches zero or more levels.

The initial state of this area is the most restrictive, using the resource names and the dot separator. By selecting the other buttons in this area you can ease the restrictions to allow more and more widgets to match the specification. The extreme case is to select all the Any Widget Chain buttons, which will match every widget in the application. As you select different buttons, the tree display will update to show you exactly which widgets will be effected by the current resource specification.

Normal and Constraint Resources

The next area allows you to select the name of the normal or constraint resources you wish to set. Some widgets may not have constraint resources, so that area will not appear.

Resource Value

This next area allows you to enter the resource value. This value should be entered exactly as you would type a line into your resource file. Thus, it should contain no unescaped newlines. There are a few special character sequences for this file:

\n This will be replaced with a newline.

\### Where # is any octal digit. This will be replaced with a single byte that contains this sequence interpreted as an octal number. For example, a value containing a NULL byte can be stored by specifying \000.

\<new-line>
This will compress to nothing.

\\ This will compress to a single backslash.

Command Area

This area contains several command buttons that I will describe in this section.

Set Save File

This button allows the user to modify the file that the resources will be saved to. This button will bring up a dialog box that will ask you for a filename; once the filename has been entered, either hit carriage return or click on the okay button. To popdown the dialog box without changing the save file, click the cancel button.

Save

This button will append the resource line described above to the end of the current save file. If no save file has been set the Set Save File dialog box will pop up to prompt the user for a filename.

Apply

This button attempts to perform an `XtSetValues` call on all widgets that match the resource line described above. The value specified is applied directly to all matching widgets. This behavior is an attempt to give a dynamic feel to the resource editor. Since this feature allows users to put an application in states it may not be willing to handle, a hook has been provided to allow specific clients to block these `SetValues` requests (see "Blocking editres Requests" below).

Unfortunately due to design constraints imposed on the widgets by the X Toolkit and the Resource Manager, trying to coerce an inherently static system into dynamic behavior can cause strange results. There is no guarantee that the results of an apply will be the same as what will happen when you save the value, and restart the application. This functionality is provided to try to give you a rough feel for what your changes will accomplish, and the results obtained should be considered suspect, at best. Having said that, this is one of the neatest features of *editres*; I strongly suggest that you play with it, and see what it can do.

Save and Apply

This button combines the Save and Apply actions described above into one button.

Popdown Resource Box

This button will remove the resource box from the display.

Blocking editres Requests

The *editres* protocol has been built into the Athena Widget set. This allows all applications that are linked against Xaw to be able to speak to the resource editor. While this provides great flexibility and can be a useful tool, it can quite easily be abused. It is therefore possible for any Xaw client to specify a value for the `editresBlock` resource described below, to keep *editres* from divulging information about its internals, or to disable the `SetValues` part of the protocol.

`editresBlock` (class `EditresBlock`)
Specifies which type of blocking this client wishes to impose on the *editres* protocol. The accepted values are:

`all`	Block all requests.
`setValues`	Block all `setValues` requests (the only *editres* request that actually modifies the application); in effect, the application is read-only.
`none`	Allow all *editres* requests.

Remember that these resources are set on any Xaw client, *not* on *editres*. They allow individual clients to keep all or some of the requests *editres* makes from ever succeeding. Of course, *editres* is also an Xaw client, so it may also be viewed and modified by *editres* (rather recursive, I know). These commands can be blocked by setting the `editresBlock` resource on *editres* itself.

Options

editres accepts all of the standard X Toolkit command-line options. For a description of the Toolkit options, see Chapter 10, *Command-line Options*.

editres accepts no application-specific options.

Note, however, that if you supply *editres* with an invalid option, you'll get the following misleading syntax message:

```
Usage: editres [ -vspace <value> ] [ -hspace <value> ]
```

This is a bug. **–vspace and –hspace are not valid options.**

Resources

editres understands all of the Core resource names and classes. (See Appendix G, *Widget Resources*, for more information.) In addition, *editres* recognizes the following application-specific resources:

`flashTime` (class `FlashTime`)
Amount of time between the flashes described above.

`flashColor` (class `FlashColor`)
Specifies the color used to flash client widgets. A bright color should be used that will immediately draw your attention to the area being flashed, such as red or yellow.

`numFlashes` (class `NumFlashes`)
Specifies the number of times the widgets in the client application will be flashed when the Show Active Widgets command is invoked.

`saveResourcesFile` (class `SaveResourcesFile`)
This is the file the resource line will be append to when the Save button is activated in the resource box.

Widgets

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *editres*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Editres  editres
        Paned  paned
                Box  box
                        MenuButton  commands
                                SimpleMenu  menu
                                SmeBSB  sendTree
                                SmeBSB  refreshTree
                                SmeBSB  dumpTreeToFile
                                SmeLine  line
                                SmeBSB  getResourceList
                                SmeLine  line
                                SmeBSB  quit
                        MenuButton  treeCommands
                                SimpleMenu  menu
                                SmeBSB  showClientWidget
                                SmeBSB  selectAll
                                SmeBSB  unselectAll
                                SmeBSB  invertAll
                                SmeLine  line
                                SmeBSB  selectChildren
                                SmeBSB  selectParent
                                SmeBSB  selectDescendants
                                SmeBSB  selectAncestors
                                SmeLine  line
                                SmeBSB  showWidgetNames
                                SmeBSB  showClassNames
                                SmeBSB  showWidgetIDs
                                SmeBSB  showWidgetWindows
                                SmeLine  line
                                SmeBSB  flashActiveWidgets
                Paned  hPane
                        Panner  panner
                        Label  userMessage
                        Grip  grip
                Porthole  porthole
                        Tree  tree
                                Toggle  <name of widget in client>
                                .
                                .
                                .
                                TransientShell  resourceBox
                                Paned  pane
                                Label  resourceLabel
```

```
                                Form  namesAndClasses
                                Toggle  dot
                                Toggle  star
                                Toggle  any
                                Toggle  name
                                Toggle  class
                                     .
                                     .
                                     .
                                Label  namesLabel
                                List  namesList
                                Label  constraintLabel
                                List  constraintList
                                Form  valueForm
                                Label  valueLabel
                                Text  valueText
                                Box  commandBox
                                Command  setFile
                                Command  save
                                Command  apply
                                Command  saveAndApply
                                Command  cancel
                                Grip  grip
                    Grip  grip
```

Files

/usr/lib/X11/app-defaults/Editres
Specifies required resources.

See Also

X, xrdb; Chapter 11, *Setting Resources*; Appendix G, *Athena Widget Resources*.

Bugs

editres prints a usage line listing two invalid options: `-vspace` and `-hspace`.

Restrictions

This is a prototype; there are lots of nifty features I would love to add, but I hope this will give you some ideas about what a resource editor can do.

Author

Chris D. Peterson, formerly of the MIT X Consortium.

Name

fs – X font server.

Syntax

fs [*options*]

Description

fs is the X Window System font server, introduced in Release 5. It supplies fonts to X Window System display servers. The font server makes it possible to use fonts on more than one host on the network.

The server is usually run by a system administrator, and started via boot files like */etc/rc.local*. Users may also wish to start private font servers for specific sets of fonts. For more information, see Chapter 6, *Font Specification*, and Volume Eight, *X Window System Administrator's Guide*.

Options

`-config` *configuration_file*
: Specifies the configuration file the font server will use.

`-ls` *listen_socket*
: Specifies a file descriptor which is already set up to be used as the listen socket. This option is only intended to be used by the font server itself when automatically spawning another copy of itself to handle additional connections.

`-port` *tcp_port*
: Specifies the TCP port number on which the server will listen for connections.

Signals

SIGTERM
: Causes the font server to exit cleanly.

SIGUSR1
: Used to cause the server to reread its configuration file.

SIGUSR2
: Used to cause the server to flush any cached data it may have.

SIGHUP Used to cause the server to reset, closing all active connections and rereading the configuration file.

Configuration File

The configuration file is a list of keyword and value pairs. Each keyword is followed by an equal sign (=) and then the desired value.

Recognized keywords include:

`catalogue` (list of string)
: Ordered list of font path element names. Use of the keyword `catalogue` is very misleading at present; the current implementation only supports a single catalogue (`all`), containing all of the specified fonts.

`alternate-servers` (list of string)
: List of alternate servers for this font server.

`client-limit` (cardinal)
: Number of clients this font server will support before refusing service. This is useful for tuning the load on each individual font server.

`clone-self` (boolean)
: Whether this font server should attempt to clone itself when it reachs the client-limit.

`default-point-size` (cardinal)
: The default pointsize (in decipoints) for fonts that don't specify.

`default-resolutions` (list of resolutions)
: Resolutions the server supports by default. This information may be used as a hint for prerendering, and substituted for scaled fonts which do not specify a resolution.

`error-file` (string)
: Filename of the error file. All warnings and errors will be logged here.

`port` (cardinal)
: TCP port on which the server will listen for connections.

`use-syslog` (boolean)
: Whether syslog(3) (on supported systems) is to be used for errors.

Example

```
#
# sample font server configuration file
#

# allow a max of 10 clients to connect to this font server
client-limit = 10

# when a font server reaches its limit, start up a new one
clone-self = on

# alternate font servers for clients to use
alternate-servers = hansen:7001,hansen:7002

# where to look for fonts
# the first is a set of Speedo outlines, the second is a set of
# misc bitmaps and the last is a set of 100dpi bitmaps
#
```

```
catalogue = /usr/lib/fonts/speedo,
    /usr/lib/X11/ncd/fonts/misc,
    /usr/lib/X11/ncd/fonts/100dpi/

# in 12 points, decipoints
default-point-size = 120

# 100 x 100 and 75 x 75
default-resolutions = 100,100,75,75
```

Font Server Names

One of the following forms can be used to name a font server that accepts TCP connections:

tcp/*hostname*:*port*
tcp/*hostname*:*port*/*catalogue_list*

The *hostname* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *port* is the decimal TCP port on which the font server is listening for connections. The *catalogue_list* specifies a list of catalogue names, with a plus sign (+) as a separator.

Examples:

```
tcp/expo.lcs.mit.edu:7000, tcp/18.30.0.212:7001/all
```

One of the following forms can be used to name a font server that accepts DECnet connections:

decnet/*nodename*::font$*object_name*
decnet/*nodename*::font$*object_name*/*catalogue_list*

The *nodename* specifies the name (or decimal numeric address) of the machine on which the font server is running. The *object_name* is a normal, case-insensitive DECnet object name. The *catalogue_list* specifies a list of catalogue names, with a plus sign (+) as a separator. Some examples follow:

```
DECnet/SRVNOD::FONT$DEFAULT, decnet/44.70::font$special/symbols
```

See Also

X; Chapter 6, *Font Specification*; Volume Eight, *X Window System Administrator's Guide*; Also see the document *Font server implementation overview* in the MIT source.

Bugs

Multiple catalogues should be supported.

Copyright

Authors

Dave Lemke, Network Computing Devices, Inc.;
Keith Packard, Massachusetts Institute of Technology.

fsinfo

— Display Font Server Info

Name

fsinfo – font server information utility.

Syntax

`fsinfo` [`-server` *server_name*]

Description

The *fsinfo* program (introduced in Release 5) is a utility for displaying information about an X font server. (See the *fs* reference page, Chapter 6, *Font Specification*, and Volume Eight, *X Window System Administrator's Guide*.) *fsinfo* is used to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the font catalogues and alternate servers that are available.

Options

`-server` *server_name*

Specifies a particular font server. The *server_name* generally has the form *transport/host:port*. If the FONTSERVER environment variable is not defined, this option must be given.

Example

The following shows a sample produced by *fsinfo*.

```
name of server:hansen:7000
version number:1
vendor string:Font Server Prototype
vendor release number:17
maximum request size:16384 longwords (65536 bytes)
number of catalogues:1
    all
Number of alternate servers: 2
    #0hansen:7001
    #1hansen:7002
number of extensions:0
```

Environment Variables

FONTSERVER

To get the default font server.

See Also

fs, fslsfonts; Chapter 6, *Font Specification*; Volume Eight, *X Window System Administrator's Guide*.

Copyright

Author

Dave Lemke, Network Computing Devices, Inc.

fslsfonts

Name

fslsfonts – list X font server fonts.

Syntax

fslsfonts [*options*] [-fn *pattern*]

Description

fslsfonts lists the fonts from a font server that match the given *pattern*.

The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

fslsfonts has been added to the standard distribution of X11 in Release 5 and is intended to be run with the X font server. (For more information, see the *fs* reference page, Chapter 6, *Font Specification*, and Volume Eight, *X Window System Administrator's Guide*.)

Options

-1 Indicates that listings should use a single column. This is the same as -n 1.

-C Indicates that listings should use multiple columns. This is the same as -n 0.

-fn *pattern*
Lists only those fonts matching the given *pattern*.

-l[l[l]]
Indicates that medium, long, and very long listings, respectively, should be generated for each font.

-m Indicates that long listings should also print the minimum and maximum bounds of each font.

-n *columns*
Specifies the number of columns to use in displaying the output. By default, it will attempt to fit as many columns of font names into the number of characters specified by -w *width*.

-server *server_name*
Specifies a particular font server. The *server_name* generally has the form *transport/host:port*. If the FONTSERVER environment variable is not defined, this option must be given.

-u Indicates that the output should be left unsorted.

-w *width*
Specifies the width in characters that should be used in figuring out how many columns to print. The default is 79.

See Also

fs, fsinfo, showfont, xlsfonts; Chapter 6, *Font Specification*; Volume Eight, *X Window System Administrator's Guide*.

Environment Variables

FONTSERVER

To get the default font server.

Bugs

Running `fslsfonts -l` can tie up your server for a very long time. This delay is really a bug with single-threaded, non-preemptable servers, not with the *fslsfonts* client.

Copyright

Author

Dave Lemke, Network Computing Devices, Inc.

fstobdf

— Font Server to BDF Converter

Name

fstobdf – Convert font server font to BDF format.

Syntax

fstobdf -fn *fontname*[-server *server_name*]

Description

The *fstobdf* program reads a font from a font server and prints a BDF file on the standard output that may be used to recreate the font. This is useful in testing servers, debugging font metrics, and reproducing lost BDF files.

fstobdf has been added to the standard distribution of X11 in Release 5 and is intended to be used with the X font server. (For more information, see the *fs* reference page and Volume Eight, *X Window System Administrator's Guide*.)

Options

-fn *fontname*

Specifies the font for which a BDF file should be generated.

-server *server_name*

Specifies a particular font server. The *server_name* generally has the form *transport/host:port*. If the FONTSERVER environment variable is not defined, this option must be given.

Environment Variables

FONTSERVER

To get the default font server.

See Also

bdftosnf, fs, fsinfo, fslsfonts; Volume Eight, *X Window System Administrator's Guide*; Appendix M, *Logical Font Description Conventions*, in Volume Zero, *X Protocol Reference Manual*. Also see the document *Bitmap Distribution Format*, in the MIT distribution.

Copyright

Authors

Olaf Brandt, Network Computing Devices.
Dave Lemke, Network Computing Devices.
Jim Fulton, MIT X Consortium.

listres

Name

listres – list resources in widgets.

Syntax

listres [*options*] [*widget . . .*]

Description

The *listres* client generates a list of each specified widget's resource database. The list includes the class in which each resource is first defined, the instance and class name, and the type of each resource.

If no widgets are specified (or the `-all` option is used), a two-column list of known widget names and their class hierarchies is printed. In the MIT distribution, this includes the intrinsics-defined widget classes Core, Composite, Constraint, and Shell (and Shell's six subclasses), plus the Athena widgets.

Case is not significant when specifying the name of the widget or widgets whose resources are to be printed. For example:

```
% listres Core
```

is equivalent to:

```
% listres CORE
```

Options

listres accepts the following application-specific options:

`-all` Indicates that *listres* should print information for all known widgets and objects.

`-format` *printf_string*
: Specifies the *printf*-style format string to be used to print out the name, instance, class, and type of each resource.

`-nosuper`
: Specifies that resources inherited from a superclass should not be listed. This is useful for determining which resources are new to a subclass.

`-top` *name*
: Specifies the name of the widget to be treated as the top of the hierarchy. Case is not significant, and the name may match either the class variable name or the class name. The default is `core`.

`-tree` Specifies that all widgets and objects be listed in a tree.

`-variable`
: Indicates that widgets should be identified by the names of the class record variables rather than the class name given in the variable. This is useful for distinguishing subclasses that have the same class name as their superclasses.

listres also *recognizes* all of the standard X Toolkit options (i.e., the program will run); however, since *listres* is not a window-based application, it does not *use* them.

Resources

`resourceFormat` (class `ResourceFormat`)
: Specifies the *printf*-style format string to be used to print out the name, instance, class, and type of each resource.

`showSuper` (class `ShowSuper`)
: If false, resources inherited from a superclass are not listed. This is useful for determining which resources are new to a subclass. The default is true.

`showTree` (class `ShowTree`)
: If true, specifies that all widgets and objects be listed in a tree. The default is false.

`showVariable` (class `ShowVariable`)
: If true, widgets are identified by the names of the class record variables rather than the class name given in the variable. This is useful for distinguishing subclasses that have the same class name as their superclasses. The default is false.

`topObject` (class `TopObject`)
: Specifies the name of the widget to be treated as the top of the hierarchy. Case is not significant, and the name may match either the class variable name or the class name. The default is `core`.

See Also

X, xrdb, appres, editres; Chapter 11, *Setting Resources*; Appendix G, *Widget Resources*; Volume Four, *X Toolkit Intrinsics Programming Manual*; Volume Five, *X Toolkit Intrinsics Reference Manual*.

Bugs

On operating systems that do not support dynamic linking of run-time routines, this program must have all of its known widgets compiled in. The sources provide several tools for automating this process for various widget sets.

Author

Jim Fulton, MIT X Consortium.

mkfontdir

Create fonts.dir Files

Name

mkfontdir – creates a *fonts.dir* file for each specified directory of font files.

Syntax

mkfontdir [*directory_names*]

Description

For each directory argument, *mkfontdir* reads all of the font files in the directory and searches for properties named "FONT", or (failing that) the name of the file stripped of its suffix. These are used as font names, which are written out to the file *fonts.dir* in the directory, along with the name of the font file.

The kinds of font files read by *mkfontdir* depend on configuration parameters, but typically include PCF (suffix *.pcf*), SNF (suffix *.snf*), and BDF (suffix *.bdf*). For more information, see Volume Eight, *X Window System Administrator's Guide*. If a font exists in multiple formats, *mkfontdir* will first choose PCF, then SNF, and finally BDF.

Scalable Fonts

Because scalable font files do not usually include the X font name, the *fonts.dir* file in directories containing such fonts must be edited by hand to include the appropriate entries for those fonts. However, be aware that when *mkfontdir* is subsequently run, all of those additions will be lost. (It might be advisable to maintain an additional list of scalable fonts that can be read into the *fonts.dir* file each time *mkfontdir* is run.)

Font Name Aliases

The file *fonts.alias*, which can be put in any directory of the font path, is used to map new names to existing fonts, and should be edited by hand. The format is straightforward enough: two white-space separated columns, the first containing aliases and the second containing font-name patterns.

When a font alias is used, the name it references is searched for in the normal manner, looking through each font directory in turn. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white space in either name, simply enclose the name in double-quote marks. To embed double-quote marks (or any other special character), precede it with a backslash:

```
"magic-alias with spaces"    "\"fontname\" with quotes"
regular-alias                fontname
```

Searching the Font Path

Both the X server and the font server (*fs*) look for *fonts.dir* and *fonts.alias* files in each directory in the font path each time the font path is set (see *xset*).

See Also

X, Xserver, fs, xset; Volume Eight, *X Window System Administrator's Guide*; Appendix M, *Logical Font Description Conventions*, in Volume Zero, *X Protocol Reference Manual*.

mwm

Motif window manager

Name

mwm – the Motif window manager.

Syntax

mwm [*options*]

Description

The Motif window manager, *mwm*, provides all of the standard window management functions. It allows you to move, resize, iconify/deiconify, maximize, and close windows and icons, focus input to a window or icon, refresh the display, etc. *mwm* provides much of its functionality via a frame that (by default) is placed around every window on the display. The *mwm* frame has the three-dimensional appearance characteristic of the OSF/Motif graphical user interface.

Chapters 3 and 4 of this guide discuss input focus, the components of the *mwm* frame, and the various window management functions you can perform using the pointer on the frame. In addition, Chapter 4 describes how to perform window/icon management functions using *mwm*'s Window Menu (as well as keyboard shortcuts for menu functions). Chapter 4 also describes the Root Menu, which provides commands that can be thought of as affecting the display as a whole.

By default, *mwm* manages only screen 0. (You can specify an alternate screen by setting the DISPLAY environment variable or using the `-display` option.) If you want *mwm* to manage all screens on the display, use the `-multiscreen` option or set the `multiScreen` resource to `True`. (See "mwm-specific Appearance and Behavior Resources.")

You can customize dozens of *mwm* features by editing a startup file (*.mwmrc*) and/or by specifying resources for the *mwm* client. You can place *mwm* resources in your regular resource file (often called *.Xresources*) in your home directory; or you can create a file called *Mwm* (also in your home directory) for *mwm* resources only. If you place conflicting specifications in both files, the resources in *.Xresources* (presuming they are loaded into the RESOURCE_MANAGER) take precedence. (These files are generally kept in the user's home directory. Note, however, that the actual location of resource files may depend on certain environment variables. See the section "Environment Variables" for more information.)

Chapter 13 of this guide describes the syntax of the *.mwmrc* file and of *mwm* resource specifications. Chapter 13 also describes how to use an icon box, which can be set up to organize icons on the display.

The current reference page primarily describes the actions and resources by *mwm*. This reference page should assist you in customizing *mwm*, according to the guidelines specified in Chapter 13.

Options

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

Specifies the name of the display on which to run *mwm*. *host* is the hostname of the physical display, *server* specifies the server number, and *screen* specifies the screen number. Either or both of the *host* and *screen* elements can be omitted. If *host* is omitted, the local display is assumed. If *screen* is omitted, screen 0 is assumed (and the period is unnecessary). The colon and (display) *server* are necessary in all cases.

For example, the following command runs *mwm* on screen 1 on server 0 on the display named *your_node*.

```
mwm -display your_node:0.1
```

`-multiscreen`

Specifies that *mwm* should manage all screens on the display. The default is to manage only screen 0. You can specify an alternate screen by setting the DISPLAY environment variable or using the `-display` option.

You can also specify that *mwm* manage all screens by assigning a value of `True` to the `multiScreen` resource variable. See "mwm-specific Appearance and Behavior Resources."

`-name app_name`

Specifies the name under which resources for the window manager should be found.

`-screens screen_name [screen_name ]...`

Assigns resource names to the screens *mwm* is managing. (By default, the screen number is used as the *screen_name*.) If *mwm* is managing a single screen, only the first name in the list is used. If *mwm* is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. If there are more screens than names, resources for the remaining screens will be retrieved using the first *screen_name*.

`-xrm resourcestring`

Specifies a resource name and value to override any defaults. This option is very useful for setting resources that don't have explicit command-line arguments.

.mwmrc Startup File

The default operation of *mwm* is largely controlled by a system-wide file, called *system.mwmrc*, which establishes the contents of the Root Menu and Window Menu, how menu functions are invoked, and what key and button combinations can be used to manage windows. To modify the behavior of *mwm*, you can edit a copy of this file in your home directory. The version of this file in your home directory should be called *.mwmrc*. (You can specify an alternate startup file using the `configFile` resource variable.)

The syntax of the *system.mwmrc* file is described in Chapter 13 of this guide. Chapter 13 also examines how to create menus, how to modify existing menus, and how to bind window manager functions to keystrokes, pointer button actions, or a combination of keys and buttons.

In describing the syntax of button and key bindings, Chapter 13 refers to the variables *modifier_key* and *button_event*. Acceptable values for *modifier_key* are: `Ctrl`, `Shift`, `Alt`, `Meta`, `Lock`, `Mod1`, `Mod2`, `Mod3`, `Mod4`, and `Mod5`. *mwm* considers `Alt` and `Meta` to be equivalent. See Chapter 14, *Setup Clients*, for a discussion of modifier keys and key mapping.

The acceptable values for *button_event* are:

```
Btn1Down
Btn1Up
Btn1Click
Btn1Click2
Btn2Down
Btn2Up
Btn2Click
Btn2Click2
    .
    .
    .
Btn5Down
Btn5Up
Btn5Click
Btn5Click2
```

Most of these button actions are obvious. (A specification ending in `Click2` refers to a double click. Thus, `Button1Click2` means to double click button 1.) Note that the list indicates a range between button 1 and button 5 (i.e., the same button events can be specified for buttons 2, 3, and 4).

mwm Functions

This section describes the functions you can specify in an *mwm* startup file.

Unless otherwise noted, you can specify that each action is invoked:

- In any of the following contexts: `root`, `window`, and `icon`.
- Using button bindings, key binding, or menu items.

When a function is specified with the context `icon | window` and you invoke the function from the icon box, the function applies to the icon box itself (rather than to any of the icons it contains).

A function is treated as `f.nop` when it is:

- Not a valid function name.
- Specified inappropriately (e.g., mapped to a button when the function cannot be invoked using button bindings).
- Invoked in an invalid context. (For example, you cannot invoke `f.minimize` on a window that is already iconified.)

See Chapter 13 for a discussion of context, bindings, and menus.

mwm recognizes the following functions:

`f.beep`
: Causes a beep from the keyboard.

`f.circle_down [icon | window]`
: Causes the window or icon on the top of the stack to be lowered to the bottom of the stack. If the `icon` argument is specified, the function applies only to icons. If the `window` argument is specified, the function applies only to windows.

  This function is invoked by the Shuffle Down item on the default Root Menu.

`f.circle_up [icon | window]`
: Causes the window or icon on the bottom of the stack to be raised to the top. If the `icon` argument is specified, the function applies only to icons. If the `window` argument is specified, the function applies only to windows.

  This function is invoked by the Shuffle Up item on the default Root Menu.

`f.exec [`*command*`]`

`! [`*command*`]`
: Executes *command* using the shell specified by the MWMSHELL environment variable. (If MWMSHELL isn't set, the command is executed using the shell specified by the SHELL environment variable; otherwise, the command is executed using */bin/sh*.)

`f.focus_color`
: Sets the colormap focus to a client window. If this function is invoked in the `root` context, the default colormap (specified by X for the screen where *mwm* is running) is installed and there is no specific client window colormap focus. For the `f.focus_color` function to work, the `colormapFocusPolicy` should be specified as `explicit`; otherwise the function is treated as `f.nop`.

`f.focus_key`
: Sets the input focus to a window or icon. For the `f.focus_key` function to work, the `keyboardFocusPolicy` should be specified as `explicit`. If `keyboardFocusPolicy` is not `explicit`, or if the function is invoked in the `root` context, it is treated as `f.nop`.

`f.kill`
: Terminates a client. Specifically, sends the WM_DELETE_WINDOW message to the selected window if the client application has requested it through the WM_PROTOCOLS property. The application is supposed to respond to the message by removing the indicated window. If the WM_SAVE_YOURSELF protocol is set up and the WM_DELETE_WINDOW protocol is not, the client is sent a message, indicating that the client needs to prepare to be terminated. If the client does not have the WM_DELETE_WINDOW or WM_SAVE_YOURSELF protocol set, the `f.kill` function

causes a client's X connection to be terminated (usually resulting in termination of the client).

This function is invoked by the Close item on the default Window Menu. The `f.kill` function can only be invoked in the contexts `window` and `icon`.

See also `quitTimeout` in the "Resources" section; see the WM_PROTOCOLS property in Volume Two, *Xlib Reference Manual*.

`f.lower` [`-`*`client`* | `within` | `freeFamily`]

Without arguments, lowers a window or icon to the bottom of the stack. (By default, the context in which the function is invoked indicates the window or icon to lower.) If an application window has one or more transient windows (e.g., dialog boxes), the transient windows are lowered with the parent (within the global stack) and remain on top of it. (The `within` and `freeFamily` arguments allow you to control how transient windows are affected by the `f.lower` action.) `f.lower` is invoked (without arguments) by the Lower item on the default Window Menu.

If the `-`*`client`* argument is specified, the function is invoked on the named client. (*`client`* must be the instance or class name of a program.)

The `within` argument is used to lower a transient window within the application's "local" window hierarchy; all transients remain above the parent window (usually the main application window) and that window remains in the same position in the global window stack. In practice, this function is only useful when there are two or more transient windows and you want to shuffle them.

The `freeFamily` argument is used to lower a transient below its parent—in effect, to free transient windows from the local hierarchy. Again, the parent is not moved in the global window stack. (Note, however, that if you use this function on the parent, the entire family stack is lowered within the global stack.)

`f.maximize`

Causes a window to be redisplayed at its maximum size. The `f.maximize` function is invoked by the Maximize item on the default Window Menu. This function cannot be invoked in the context `root` or on a window that is already maximized.

`f.menu` *`menu_name`*

Associates a cascading (i.e., pull-right) menu with a menu item (from which the cascading menu is displayed); or associates a menu with a button or key binding. The *`menu_name`* argument specifies the menu.

`f.minimize`

Causes a window to be minimized (i.e., iconified). When no icon box is being used, icons are placed on the bottom of the stack (generally in the lower-left corner of the screen. See also `iconPlacement` in "mwm-specific Appearance and Behavior Resources)." If an icon box is being used, icons are placed inside the box.

The f.minimize function is invoked by the Minimize item on the default Window Menu. This function cannot be invoked in the context root or on an iconified window.

f.move
Allows you to move a window interactively, using the pointer. This function is invoked by the Move item on the default Window Menu.

f.next_cmap
Installs the next colormap in the list of colormaps for the window with the colormap focus. (See f.focus_color.)

f.next_key [icon | window | transient]
Without any arguments, this function advances the input focus to the next window or icon in the stack. You can specify only icon or window to make the function apply only to icons or windows, respectively.

Generally, the focus is moved only to windows that do not have an associated secondary window that is application modal. (An active dialog box is application modal.) If the transient argument is specified, transient (secondary) windows are also traversed. Otherwise, if only window is specified, focus is moved only to the last window in a transient group to have the focus.

For this function to work, keyboardFocusPolicy must be explicit; otherwise, the function is treated as f.nop. See Chapter 4 for the default key combinations to move the focus.

f.nop Specifies no operation. (In other words, it does nothing.)

f.normalize
Causes a client window to be displayed at its normal size. The f.normalize function is invoked by the Restore item on the default Window Menu. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.normalize_and_raise
Causes the client window to be displayed at its normal size and raised to the top of the stack. This function cannot be invoked in the context root or on a window that is already at its normal size.

f.pack_icons
Rearranges icons in an optimal fashion (based on the layout policy being used), either on the root window or in the icon box. See iconPlacement in "mwm-specific Appearance and Behavior Resources." (See Chapter 13 for instructions on using an icon box.)

f.pass_keys
Toggles processing of key bindings for window manager functions. When key binding processing is disabled, all keys are passed to the window with the keyboard input focus and no window manager functions are invoked. If the f.pass_keys function

is set up to be invoked with a key binding, the binding can be used to toggle (enable/disable) key binding processing.

`f.post_wmenu`

Displays the Window Menu. If a key is used to display the menu and a Window Menu command button is present, the upper-left corner of the menu is placed at the lower-left corner of the command button. If no Window Menu command button is present, the menu is placed in the upper-left corner of the window.

`f.prev_cmap`

This function installs the previous colormap in the list of colormaps for the window with the colormap focus. (See `f.focus_color`.)

`f.prev_key [icon | window | transient]`

Without any arguments, this function moves the input focus to the previous window or icon in the stack. You can specify only `icon` or `window` to make the function apply only to icons or windows, respectively.

Generally, the focus is moved only to windows that do not have an associated secondary window that is application modal. (An active dialog box is application modal.) If the `transient` argument is specified, transient (secondary) windows are also traversed. Otherwise, if only `window` is specified, focus is moved only to the last window in a transient group to have the focus.

For this function to work, `keyboardFocusPolicy` must be `explicit`; otherwise, the function is treated as `f.nop`. See Chapter 4 for the default key combinations to move the focus.

`f.quit_mwm`

Stops the *mwm* window manager. Note that this function does not stop the X server. This function cannot be invoked from a non-root menu.

`f.raise [-`*`client`*` | within | freeFamily ]`

Raises a window or icon to the top of the stack. By default, the context in which the function is invoked indicates the window or icon to raise. If the `-`*`client`* argument is specified, the function is invoked on the named client. (*`client`* must be the instance or class name of a program.)

Without arguments, raises a window or icon to the top of the stack. (By default, the context in which the function is invoked indicates the window or icon to raise.) If an application window has one or more transient windows (e.g., dialog boxes), the transient windows are raised with the parent (within the global stack) and remain on top of it. (The `within` and `freeFamily` arguments allow you to control how transient windows are affected by the `f.raise` action.)

If the `-`*`client`* argument is specified, the function is invoked on the named client. (*`client`* must be the instance or class name of a program.)

The `within` argument is used to raise a transient window within the application's "local" window hierarchy; all transients remain above the parent window (usually the main application window) and that window remains in the same position in the global window stack. In practice, this function is only useful when there are two or more transient windows and you want to shuffle them.

The `freeFamily` argument raises a transient to the top of the family stack (in effect, transient windows are freed from the local hierarchy) and also raises the parent window to the top of the global stack.

`f.raise_lower [within | freeFamily ]`

Raises a primary application window to the top of the stack or lowers a window to the bottom of the stack, as appropriate to the context.

The `within` argument is intended to raise a transient window within the application's "local" window hierarchy (if the transient is obscured); all transients remain above the parent window (usually the main application window); the parent window should also remain in the same position in the global window stack. If the transient is not obscured by another window in the local stack, the transient window is lowered within the family.

The preceding paragraph describes how `f.raise_lower within` *should* work. However, we have found that the parent window does not always remain in the same position in the global window stack.

The `freeFamily` argument raises a transient to the top of the family stack (in effect, transient windows are freed from the local hierarchy) and also raises the parent window to the top of the global stack. If the transient is not obscured by another window, this function lowers the transient to the bottom of the family stack and lowers the family in the global stack.

`f.refresh`

Redraws all windows. This function is invoked by the Refresh item on the default Root Menu.

`f.refresh_win`

Redraws a single window.

`f.resize`

Allows you to resize a window interactively, using the pointer. This function is invoked by the Size item on the default Window Menu.

`f.restart`

Restarts the *mwm* window manager. (Specifically, this function causes the current *mwm* process to be stopped and a new *mwm* process to be started.) This function is invoked by the Restart... item on the default Root Menu. It cannot be invoked from a non-root menu.

`f.restore`

Causes the client window to be displayed at its previous size. If invoked on an icon, `f.restore` causes the icon to be converted back to a window at its previous size. Thus, if the window was maximized, it is restored to this state. If the window was previously at its normal size, it is restored to this state. If invoked on a maximized window, the window is restored to its normal size. The `f.restore` function is invoked by the Restore item on the default Window Menu. This function cannot be invoked in the context `root` or on a window that is already at its normal size.

`f.restore_and_raise`

Causes the client window to be displayed at its previous size and raised to the top of the stack. This function cannot be invoked in the context `root` or on a window that is already at its normal size.

`f.screen` [`next` | `prev` | `back` | *`screen_number`*]

Causes the pointer to be warped to (i.e., redrawn at) another screen, determined by one of four mutually exclusive parameters:

The `next` argument means skip to the next managed screen. The `prev` argument means skip back to the previous managed screen. The `back` argument means skip to the last screen visited (regardless of its position in the numeric order).

Screens are normally numbered beginning at 0. *`screen_number`* specifies a particular screen.

`f.send_msg` *`message_number`*

Sends a message of the type _MOTIF_WM_MESSAGES to a client; the message type is indicated by the *`message_number`* argument. The message is sent only if the client's _MOTIF_WM_MESSAGES property includes *`message_number`*.

If a menu item is set up to invoke `f.send_msg` and the *`message_number`* is not included in the client's _MOTIF_WM_MESSAGES property, the menu item label is greyed out (indicating that it is not available for selection).

`f.separator`

Creates a divider line in a menu. Any associated label is ignored.

`f.set_behavior`

Restarts *mwm*, toggling between the default behavior for the particular system and the user's custom environment. In any case, a dialog box asks the user to confirm or cancel the action. By default this function is invoked using the following key sequence: `Shift Ctrl Meta !`.

`f.title`

Specifies the title of a menu. The title string is separated from the menu items by a double divider line.

Resources

mwm resources are considered to fall into three categories:

- *mwm* component appearance resources. These resources set the characteristics of *mwm*'s component features, such as the window frame, menus, and icons.
- *mwm*-specific appearance and behavior resources. These resources set characteristics of the window manager client, such as focus policy, key and button bindings, and so forth.
- Client-specific resources. These *mwm* resources can be used to set the appearance and behavior of a particular client or class of clients.

The following sections simply describe the valid resources. For a discussion of *mwm* resource syntax, see Chapter 13 in this guide. (For more information about basic resource syntax and the precedence of resource specifications, see Chapter 11.)

Note that `Mwm` is the class name. You can specify resources for multiple screens using the names supplied to the `-screens` command-line option in place of `mwm` or `Mwm` in the resource line. (See "Options.")

mwm Component Appearance Resources

The Motif window manager can be considered to be made up of the following components: client window frames, menus, icons, and feedback (dialog) boxes. Some component appearance resources can be set for all of these components; others can be set only for the frame and icons.

Unless a default is specified, the default varies based on system specifics (such as screen type, color resources, etc.).

The following component appearance resources apply to all window manager components:

`background` (class `Background`)
: Specifies the background color.

`backgroundPixmap` (class `BackgroundPixmap`)
: Specifies the background pixmap of the *mwm* decoration when the window does not have the input focus (i.e., is inactive).

`bottomShadowColor` (class `Foreground`)
: Specifies the color to be used for the lower and right bevels of the window manager decoration.

`bottomShadowPixmap` (class `BottomShadowPixmap`)
: Specifies the pixmap to be used for the lower and right bevels of the window manager decoration.

`fontList` (class `FontList`)
: Specifies the font to be used in the window manager decoration. The default is `fixed`.

`foreground` (class `Foreground`)
: Specifies the foreground color.

`saveUnder` (class `SaveUnder`)
Specifies whether save unders are used for *mwm* components. By default (`False`), save unders will not be used on any window manager frames.

Save unders must be implemented by the X server for this function to to take effect. When save unders are implemented, the X server saves the contents of windows obscured by other windows that have the save under attribute set. If the `saveUnder` resource is `True`, *mwm* will set the save under attribute on the frame of any client that has it set.

`topShadowColor` (class `Background`)
Specifies the color to be used for the upper and left bevels of the window manager decoration.

`topShadowPixmap` (class `TopShadowPixmap`)
Specifies the pixmap to be used for the upper and left bevels of the window manager decoration.

The following component appearance resources apply to the window frame and icons:

`activeBackground` (class `Background`)
Specifies the background color of the *mwm* decoration when the window has the input focus (i.e., is active).

`activeBackgroundPixmap` (class `ActiveBackgroundPixmap`)
Specifies the background pixmap of the *mwm* decoration when the window has the input focus (i.e., is active).

`activeBottomShadowColor` (class `Foreground`)
Specifies the bottom shadow color of the *mwm* decoration when the window has the input focus (i.e., is active).

`activeBottomShadowPixmap` (class `BottomShadowPixmap`)
Specifies the bottom shadow pixmap of the *mwm* decoration when the window has the input focus (i.e., is active).

`activeForeground` (class `Foreground`)
Specifies the foreground color of the *mwm* decoration when the window has the input focus (i.e., is active).

`activeTopShadowColor` (class `Background`)
Specifies the top shadow color of the *mwm* decoration when the window has the input focus (i.e., is active).

`activeTopShadowPixmap` (class `TopShadowPixmap`)
Specifies the top shadow Pixmap of the *mwm* decoration when the window has the input focus (i.e., is active).

mwm-specific Appearance and Behavior Resources

The *mwm*-specific resources control aspects of what you probably think of as the window manager application itself, features such as the focus policy, whether windows are placed on

the display automatically or interactively, which set(s) of button and key bindings are used, whether an icon box is used, and so forth.

The following *mwm*-specific appearance and behavior resources can be specified:

`autoKeyFocus` (class `AutoKeyFocus`)
: If `True` (the default), when the focus window is withdrawn from window management or is iconified, the focus bounces back to the window that previously had the focus. This resource is available only when `keyboardFocusPolicy` is `explicit`. If `False`, the input focus is not set automatically. `autoKeyFocus` and `startupKeyFocus` should both be `True` to work properly with tear off menus.

`autoRaiseDelay` (class `AutoRaiseDelay`)
: Specifies the amount of time (in milliseconds) that *mwm* will wait before raising a window after it receives the input focus. The default is `500`. This resource is available only when `focusAutoRaise` is `True` and the `keyboardFocusPolicy` is `pointer`.

`bitmapDirectory` (class `BitmapDirectory`)
: Identifies the directory to be searched for bitmaps referenced by *mwm* resources (if an absolute pathname to the bitmap file is not given). The default is */usr/include/X11/bitmaps*, which is considered the standard location on many systems. Note, however, that the location of the bitmap directory may vary in different environments. If a bitmap is not found in the specified directory, XBMLANGPATH is searched.

`buttonBindings` (class `ButtonBindings`)
: Identifies the set of button bindings to be used for window management functions; must correspond to a set of button bindings specified in the *mwm* startup file. Button bindings specified in the startup file are merged with built-in default bindings. The default is `DefaultButtonBindings`.

`cleanText` (class `CleanText`)
: Specifies whether text that appears in *mwm* title and feedback windows is displayed over the existing background pattern. If `True` (the default), text is drawn with a clear (no stipple) background. (Only the stippling in the area immediately around the text is cleared.) This enhances readability, especially on monochrome systems where a `backgroundPixmap` is specified. If `False`, text is drawn on top of the existing background.

`clientAutoPlace` (class `ClientAutoPlace`)
: Specifies the location of a window when the user has not specified a location. If `True` (the default), windows are positioned with the upper-left corners of the frames offset horizontally and vertically (so that no two windows completely overlap).

  If `False`, the currently configured position of the window is used.

  In either case, *mwm* will attempt to place the windows totally on screen.

`colormapFocusPolicy` (class `ColormapFocusPolicy`)
: Specifies the colormap focus policy. Takes three possible values: `keyboard`, `pointer`, and `explicit`. If `keyboard` (the default) is specified, the input focus window has the colormap focus. If `explicit` is specified, a colormap selection action is done on a client window to set the colormap focus to that window. If `pointer` is specified, the client window containing the pointer has the colormap focus.

`configFile` (class `ConfigFile`)
: Specifies the pathname for the *mwm* startup file. The default startup file is *.mwmrc*.

mwm searches for the configuration file in the user's home directory. If the `configFile` resource is not specified or the file does not exist, *mwm* defaults to an implementation-specific standard directory (the default is */usr/lib/X11/system.mwmrc*).

If the LANG environment variable is set, *mwm* looks for the configuration file in a *$LANG* subdirectory first. For example, if the LANG environment variable is set to Fr (for French), *mwm* searches for the configuration file in the directory *$HOME/Fr* before it looks in *$HOME*. Similarly, if the `configFile` resource is not specified or the file does not exist, *mwm* defaults to */usr/lib/X11/$LANG/system.mwmrc* before it reads */usr/lib/X11/system.mwmrc*.

If the `configFile` pathname does not begin with "~/", *mwm* considers it to be relative to the current working directory.

`deiconifyKeyFocus` (class `DeiconifyKeyFocus`)
: If `True` (the default), a window receives the input focus when it is normalized (deiconified). This resource applies only when the `keyboardFocusPolicy` is `explicit`.

`doubleClickTime` (class `DoubleClickTime`)
: Specifies the maximum time (in milliseconds) between the two clicks of a double click. The default is the display's multi-click time.

`enableWarp` (class `EnableWarp`)
: If `True` (the default), causes *mwm* to *warp* the pointer to the center of the selected window during resize and move operations invoked using keyboard accelerators. (The cursor symbol disappears from its current location and reappears at the center of the window.) If `False`, *mwm* leaves the pointer at its original place on the screen, unless the user explicitly moves it.

`enforceKeyFocus` (class `EnforceKeyFocus`)
: If `True` (the default), the input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scrollbar that can be operated without setting the focus to that client.) If the resource is `False`, the keyboard input focus is not explicitly set to globally active windows.

`fadeNormalIcon` (class `FadeNormalIcon`)
If `True`, an icon is greyed out when it has been normalized. The default is `False`.

`feedbackGeometry` (class `FeedbackGeometry`)
Specifies the position of the small, rectangular feedback box that displays coordinate and size information during move and resize operations. By default, the feedback window appears in the center of the screen. This resource takes the argument:

`[=]`*±xoffset±yoffset*

With the exception of the optional leading equal sign, this string is identical to the second portion of the standard geometry string. See the section "Window Geometry: Specifying Size and Location" in Chapter 3, *Working in the X Environment*, for more information. Note that `feedbackGeometry` allows you to specify location only. The size of the feedback window is not configurable using this resource. Available as of *mwm* version 1.2.

`frameBorderWidth` (class `FrameBorderWidth`)
Specifies the width in pixels of a window frame border, without resize handles. (The border width includes the three-dimensional shadows.) The default is determined according to screen specifics.

`iconAutoPlace` (class `IconAutoPlace`)
Specifies whether the window manager arranges icons in a particular area of the screen or places each icon where the window was when it was iconified. If `True` (the default), icons are arranged in a particular area of the screen, determined by the `iconPlacement` resource. If `False`, an icon is placed at the location of the window when it is iconified.

`iconBoxGeometry` (class `IconBoxGeometry`)
Specifies the initial position and size of the icon box. Takes as its argument the standard geometry string:

widthxheight±xoff±yoff

where *width* and *height* are measured in icons. The default geometry string is `6x1+0-0`, which places an icon box six icons wide by one icon high in the lower-left corner of the screen.

You can omit either the dimensions or the x and y offsets from the geometry string and the defaults apply. If the offsets are not provided, the `iconPlacement` resource is used to determine the initial placement.

The actual screen size of the icon box depends on the `iconImageMaximum` and `iconDecoration` resources, which specify icon size and padding. The default value for size is (6 × *icon_width* + padding) wide by (1 × *icon_height* + padding) high.

`iconBoxName` (class `IconBoxName`)
: Specifies the name under which icon box resources are to be found. The default is `iconbox`.

`iconBoxSBDisplayPolicy` (class `IconBoxSBDisplayPolicy`)
: Specifies what scrollbars are displayed in the icon box. The resource has three possible values: `all`, `vertical`, and `horizontal`. If `all` is specified (the default), both vertical and horizontal scrollbars are displayed at all times. `vertical` specifies that a single vertical scrollbar is displayed (this also sets the orientation of the icon box to horizontal—regardless of the `iconBoxGeometry` specification). `horizontal` specifies that a single horizontal scrollbar is displayed in the icon box (this also sets the orientation of the icon box to vertical—regardless of the `iconBoxGeometry` specification).

`iconBoxTitle` (class `IconBoxTitle`)
: Specifies the name to be used in the title area of the icon box. The default is `Icons`.

`iconClick` (class `IconClick`)
: If `True` (the default), the Window Menu is displayed when the pointer is clicked on an icon.

`iconDecoration` (class `IconDecoration`)
: Specifies how much icon decoration is used. The resource value takes four possible values (multiple values can also be supplied): `label`, which specifies that only the label is displayed; `image`, which specifies that only the image is displayed; and `activelabel`, which specifies that a label (not truncated to the width of the icon) is used when the icon has the focus.

  The default decoration for icons in an icon box is `label image`, which specifies that both the label and image parts are displayed. The default decoration for individual icons on the screen proper is `activelabel label image`.

`iconImageMaximum` (class `IconImageMaximum`)
: Specifies the maximum size of the icon image. Takes a value of *`widthxheight`* (e.g., `80x80`). The maximum size supported is 128 × 128. The default is `50x50`.

`iconImageMinimum` (class `IconImageMinimum`)
: Specifies the minimum size of the icon image. Takes a value of *`widthxheight`* (e.g., `36x48`). The minimum size supported is 16 × 16 (which is also the default).

`iconPlacement` (class `IconPlacement`)
: Specifies an icon placement scheme. Note that this resource is only useful when `useIconBox` is `False` (the default). The `iconPlacement` resource takes a value of the syntax:

  `primary_layout secondary_layout` [tight]

There are four possible layout policies:

`top`, which specifies that icons are placed from the top of the screen to the bottom; `bottom`, which specifies a bottom-to-top arrangement; `left`, which specifies that icons are placed from the left to the right; `right`, which specifies a right-to-left arrangement.

The *primary_layout* specifies whether icons are placed in a row or a column and the direction of placement. The *secondary_layout* specifies where to place new rows or columns. For example, a value of `top right` specifies that icons should be placed from top to bottom on the screen and that columns should be added from right to left on the screen.

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout*. For example, don't use `top` for the *primary_layout* and `bottom` for the *secondary_layout*.

The default placement is `left bottom` (i.e., icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows are added from the bottom of the screen to the top of the screen).

The optional argument `tight` specifies that there is no space between icons.

`iconPlacementMargin` (class `IconPlacementMargin`)
Sets the distance from the edge of the screen at which icons are placed. (The value should be greater than or equal to 0. A default value is used if an invalid distance is specified.) The default value is equal to the space between icons as they are placed on the screen (which is is based on maximizing the number of icons in each row and column).

`interactivePlacement` (class `InteractivePlacement`)
If `True`, specifies that new windows are to be placed interactively on the screen using the pointer. When a client is run, the pointer shape changes to an upper-left corner cursor; move the pointer to the location you want the window to appear and click the first button; the window is displayed in the selected location. If `False` (the default), windows are placed according to the initial window configuration attributes.

`keyBindings` (class `KeyBindings`)
Identifies the set of key bindings to be used for window management functions; must correspond to a set of key bindings specified in the *mwm* startup file. Note that key bindings specified in the startup file replace the built-in default bindings. The default is `DefaultKeyBindings`.

`keyboardFocusPolicy` (class `KeyboardFocusPolicy`)
If `explicit` focus is specified (the default), placing the pointer on a window (including the frame) or icon and pressing the first pointer button focuses keyboard input on the client. If `pointer` is specified, the keyboard input focus is directed to the client window on which the pointer rests (the pointer can also rest on the frame).

`limitResize` (class `LimitResize`)
If `True` (the default), the user is not allowed to resize a window to greater than the maximum size.

`lowerOnIconify` (class `LowerOnIconify`)
If `True` (the default), a window's icon is placed on the bottom of the stack when the window is iconified. If `False`, the icon is placed in the stacking order at the same place as its associated window.

`maximumMaximumSize` (class `MaximumMaximumSize`)
Specifies the maximum size of a client window (as set by the user or client). Takes a value of *`widthxheight`* (e.g., `1024x1024`) where *`width`* and *`height`* are in pixels. The default is twice the screen width and height.

`moveThreshold` (class `MoveThreshold`)
Controls the sensitivity of dragging operations (such as those used to move windows and icons on the display). Takes a value of the number of pixels that the pointing device is moved while a button is held down before the move operation is initiated. The default is `4`. This resource helps prevent a window or icon from moving when you click or double click and inadvertently jostle the pointer while a button is down.

`moveOpaque` (class `MoveOpaque`)
If `False` (the default), when you move a window or icon, its outline is moved before it is redrawn in the new location. If `True`, the actual (and thus, opaque) window or icon is moved. Available as of *mwm* version 1.2.

`multiScreen` (class `MultiScreen`)
If `False` (the default), *mwm* manages only a single screen. If `True`, *mwm* manages all screens on the display. (See "Options.")

`passButtons` (class `PassButtons`)
Specifies whether button press events are passed to clients after the events are used to invoke a window manager function (in the client context). If `False` (the default), button presses are not passed to the client. If `True`, button presses are passed to the client. (Note that the window manager function is done in either case.)

`passSelectButton` (class `PassSelectButton`)
Specifies whether select button press events are passed to clients after the events are used to invoke a window manager function (in the client context). If `True` (the default), button presses are passed to the client window. If `False`, button presses are not passed to the client. (Note that the window manager function is done in either case.)

`positionIsFrame` (class `PositionIsFrame`)
Specifies how *mwm* should interpret window position information (from the WM_NORMAL_HINTS property and from configuration requests). If `True` (the default), the information is interpreted as the position of the *mwm* client window

frame. If `False`, it is interpreted as being the position of the client area of the window.

`positionOnScreen` (class `PositionOnScreen`)
If `True` (the default), specifies that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen. (If a window is larger than the size of the screen, at least the upper-left corner of the window is placed is on the screen.) If `False`, windows are placed in the requested position even if totally off the screen.

`quitTimeout` (class `QuitTimeout`)
Specifies the amount of time (in milliseconds) that *mwm* will wait for a client to update the WM_COMMAND property after *mwm* has sent the WM_SAVE_YOURSELF message. The default is `1000`. (See the `f.kill` function for additional information.)

`raiseKeyFocus` (class `RaiseKeyFocus`)
If `True`, specifies that a window raised by means of the `f.normalize_and_raise` function also receives the input focus. This function is available only when the `keyboardFocusPolicy` is `explicit`. The default is `False`.

`resizeBorderWidth` (class `ResizeBorderWidth`)
Specifies the width in pixels of a window frame border, with resize handles. (The border width includes the three-dimensional shadows.) The default is determined according to screen specifics.

`resizeCursors` (class `ResizeCursors`)
If `True` (the default), the resize cursors are always displayed when the pointer is in the window resize border.

`screens` (class `Screens`)
Assigns resource names to the screens *mwm* is managing. If *mwm* is managing a single screen, only the first name in the list is used. If *mwm* is managing multiple screens, the names are assigned to the screens in order, starting with screen 0. (See also "Options.")

`showFeedback` (class `ShowFeedback`)
Specifies whether *mwm* feedback windows and confirmation dialog boxes are displayed. (Feedback windows are used to display: window coordinates during interactive placement and subsequent moves; and dimensions during resize operations. A typical confirmation dialog is the window displayed to allow the user to allow or cancel a window manager restart operation.)

`showFeedback` accepts a list of options, each of which corresponds to the type of feedback given in a particular circumstance. Depending on the syntax in which the options are entered, you can either enable or disable a feedback option (as explained later).

The possible feedback options are: `all`, which specifies that *mwm* show all types of feedback (this is the default); `behavior`, which specifies that feedback is displayed

to confirm a behavior switch; `kill`, which specifies that feedback is displayed on receipt of a KILL signal; `move`, which specifies that a box containing the coordinates of a window or icon is displayed during a move operation; `placement`, which specifies that a box containing the position and size of a window is displayed during initial (interactive) placement; `quit`, which specifies that a dialog box is displayed so that the user can confirm (or cancel) the procedure to quit *mwm*; `resize`, which specifies that a box containing the window size is displayed during a resize operation; `restart`, which displays a dialog box so that the user can confirm (or cancel) an *mwm* restart procedure; the `none` option specifies that no feedback is shown.

By default, *mwm* supplies feedback in `all` cases. (`all` encompasses all of the other options—with the exception of the `none` option.)

To limit feedback to particular cases, you can use one of two syntaxes: with the first syntax, you disable feedback in specified cases (all other default feedback is still used); with the second syntax, you enable feedback only in specified cases.

Initially, the syntax may be confusing, but it is actually quite simple. You supply this resource with a list of options to be enabled or disabled. If the first item is preceded by a minus sign, feedback is disabled for all options in the list. (Any option not listed remains enabled.)

If the first item is preceded by a plus sign (or no sign is used), feedback is enabled only for options in the list.

For example, the following resource specification:

```
Mwm*showFeedback: resize placement restart
```

enables the feedback options `resize`, `placement`, and `restart`. What this means is the following: size information is displayed when a window is resized; coordinates are displayed when a window is placed interactively on the screen; and a dialog box is displayed so that the user can confirm or cancel a window manager restart request. Feedback is supplied in these circumstances only, overriding the default `all` option.

The following line specifies the same characteristics using the alternate syntax:

```
Mwm*showFeedback: -move kill behavior quit
```

This line disables the feedback options `move`, `kill`, `behavior`, and `quit`. The other options encompassed by the default `all` (`resize`, `placement`, and `restart`) remain enabled.

`startupKeyFocus` (class `StartupKeyFocus`)

If `True` (the default), the input focus is transferred to a window when the window is mapped (i.e., initially managed by the window manager). This function is available only when `keyboardFocusPolicy` is `explicit`. `startupKeyFocus` and `autoKeyFocus` should both be `True` to work properly with tear off menus.

`transientDecoration` (class `TransientDecoration`)
Specifies the amount of decoration *mwm* puts on transient windows. The decoration specification is exactly the same as for the `clientDecoration` (client-specific) resource. Transient windows are identified by the WM_TRANSIENT_FOR property, which is added by the client to indicate a relatively temporary window. The default is `menu title`, which specifies that transient windows have resize borders and a titlebar with a Window Menu command button.

If the client application also specifies which decorations the window manager should provide, *mwm* uses only those features that both the client and the `transient-Decoration` resource specify.

`transientFunctions` (class `TransientFunctions`)
Specifies which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the `clientFunctions` (client-specific) resource. The default is `-minimize maximize`.

If the client application also specifies which window management functions should be applicable, *mwm* provides only those functions that both the client and the `transientFunctions` resource specify.

`useIconBox` (class `UseIconBox`)
If `True`, icons are placed in an icon box. By default, the individual icons are placed on the root window.

`wMenuButtonClick` (class `WMenuButtonClick`)
If `True` (the default), a pointer button click on the Window Menu button displays the Window Menu and leaves it displayed.

`wMenuButtonClick2` (class `WMenuButtonClick2`)
If `True`, double clicking on the Window Menu command button removes the client window (actually invokes the `f.kill` function).

Client-specific Resources

Some *mwm* resources can be set to apply to certain client applications or classes of applications. Many of the client-specific resources provide what might be considered advanced customization.

The following client-specific resources can be specified:

`clientDecoration` (class `ClientDecoration`)
Specifies the amount of window frame decoration. The default frame is composed of several component parts: the titlebar, resize handles, border, and three command buttons (Minimize, Maximize, and Window Menu). You can limit the frame decoration for a client using the `clientDecoration` resource.

`clientDecoration` accepts a list of options, each of which corresponds to a part of the client frame. Depending on the syntax in which the options are entered, you can either enable or disable an option (as explained later).

The options are: `maximize` (button); `minimize` (button); `menu` (the Window Menu button); `border`; `title` (titlebar); `resizeh` (resize handles); `all`, which encompasses all decorations previously listed (this is the default); and `none`, which specifies that no decorations are used.

Some decorations require the presence of others; if you specify such a decoration, any decoration required with it will be used automatically. Specifically, if any of the command buttons is specified, a titlebar is also used; if resize handles or a titlebar is specified, a border is also used.

By default, a client window has `all` decoration. To specify only certain parts of the default frame, you can use one of two syntaxes: with the first syntax, you disable certain frame features (all other default features are still used); with the second syntax, you enable only certain features. (The syntax is virtually the same as that described for `showFeedback`.)

You supply `clientDecoration` with a list of options to be enabled or disabled. If the first item is preceded by a minus sign, the features in the list are disabled. (Any option not listed remains enabled.)

If the first item is preceded by a plus sign (or no sign is used), only those features listed are enabled.

For example, the following resource specification:

```
Mwm*XCalc*clientDecoration: -minimize maximize menu
```

removes the three command buttons from *xcalc* window frames. (The window will still have the titlebar, resize handles, and border.)

The following line specifies the same characteristics using the alternate syntax:

```
Mwm*XCalc*clientDecoration: title resizeh border
```

`clientFunctions` (class `ClientFunctions`)

Specifies whether certain *mwm* functions can be invoked on a client window. (See "mwm Functions" earlier in this reference page.) The only functions that can be controlled are those that are executable using the pointer on the default window frame.

`clientFunctions` accepts a list of options, each of which corresponds to an *mwm* function. Depending on the syntax in which the options are entered, you can either allow or disallow a function (as explained later).

The options (and the functions to which they correspond) are: `resize` (`f.resize`); `move` (`f.move`); `minimize` (`f.minimize`); `maximize` (`f.max-`

imize); close (f.kill); all (encompasses all of the previously listed functions); none (none of the default functions is allowed).

By default, a client recognizes all functions. To limit the functions a client recognizes, you can use one of two syntaxes: with the first syntax, you disallow certain functions (all other default functions are still allowed); with the second syntax, you allow only *certain* functions. (The syntax is virtually the same as that described for clientDecoration.)

You supply clientFunctions with a list of options (corresponding to functions) to be allowed or disallowed. If the first item is preceded by a minus sign, the functions in the list are disallowed. (Any option not listed remains allowed.)

If the first option is preceded by a plus sign (or no sign is used), only those functions listed are allowed.

A less than obvious reprecussion of disallowing a particular function is that the client window frame will be altered to prevent your invoking that function. For instance, if you disallow the f.resize function for a client, the client's frame will not include resize borders. (Features of the frame can also be suppressed using the clientDecoration resource.) In addition, the Window Menu Size item, which invokes the f.resize function, will no longer appear on the menu. (You cannot affect the Window Menu using clientDecoration.)

For example, the following resource line:

```
Mwm*xfd*clientFunctions: -resize maximize
```

specifies that you cannot invoke the f.resize and f.maximize functions on an *xfd* window. As a result, the resize borders and Maximize command button will not appear on the frame, and the Size and Maximize items will not appear on the Window Menu.

The following line specifies the same characteristics using the alternate syntax:

```
Mwm*xfd*clientFunctions: minimize move close
```

focusAutoRaise (class FocusAutoRaise)

If True, a window is raised when it receives the input focus. Otherwise, directing focus to a window does not affect the stacking order.

The default depends on the value assigned to the keyboardFocusPolicy resource. If the keyboardFocusPolicy is explicit, the default for focusAutoRaise is True. If the keyboardFocusPolicy is pointer, the default for focusAutoRaise is False.

If the client application also specifies which window management functions should be applicable, *mwm* provides only those functions that both the client and the clientFunctions resource specify.

`iconImage` (class `IconImage`)
: Specifies the pathname of a bitmap file to be used as an icon image for a client. (For example, you might specify: `Mwm*xclock*iconImage: ~/bitmaps/big-ben`.) The default is to display an icon image supplied by the window manager.

  If the `useClientIcon` resource is set to `True`, an icon image supplied by the client takes precedence over an icon image supplied by the user.

`iconImageBackground` (class `Background`)
: Specifies the background color of the icon image. The default is the color specified by `Mwm*background` or `Mwm*icon*background`.

`iconImageBottomShadowColor` (class `Foreground`)
: Specifies the bottom shadow color of the icon image. The default is the color specified by `Mwm*icon*bottomShadowColor`.

`iconImageBottomShadowPixmap` (class `BottomShadowPixmap`)
: Specifies the bottom shadow pixmap of the icon image. The default is the pixmap specified by `Mwm*icon*bottomShadowPixmap`.

`iconImageForeground` (class `Foreground`)
: Specifies the foreground color of the icon image. The default varies based on the icon background.

`iconImageTopShadowColor` (class `Background`)
: Specifies the top shadow color of the icon image. The default is the color specified by `Mwm*icon*topShadowColor`.

`iconImageTopShadowPixmap` (class `TopShadowPixmap`)
: Specifies the top shadow Pixmap of the icon image. The default is the pixmap specified by `Mwm*icon*topShadowPixmap`.

`matteBackground` (class `Background`)
: Specifies the background color of the matte. The default is the color specified by `Mwm*background` or `Mwm*client*background`. This resource is only relevant if `matteWidth` is positive.

`matteBottomShadowColor` (class `Foreground`)
: Specifies the bottom shadow color of the matte. The default is the color specified by `Mwm*bottomShadowColor` or `Mwm*client*bottomShadowColor`. This resource is only relevant if `matteWidth` is positive.

`matteBottomShadowPixmap` (class `BottomShadowPixmap`)
: Specifies the bottom shadow pixmap of the matte. The default is the pixmap specified by `Mwm*bottomShadowPixmap` or `Mwm*client*bottomShadowPixmap`. This resource is only relevant if `matteWidth` is positive.

matteForeground (class Foreground)
: Specifies the foreground color of the matte. The default is the color specified by Mwm*foreground or Mwm*client*foreground. This resource is only relevant if matteWidth is positive.

matteTopShadowColor (class Background)
: Specifies the top shadow color of the matte. The default is the color specified by Mwm*topShadowColor or Mwm*client*topShadowColor. This resource is only relevant if matteWidth is positive.

matteTopShadowPixmap (class TopShadowPixmap)
: Specifies the top shadow pixmap of the matte. The default is the pixmap specified by Mwm*topShadowPixmap or Mwm*client*topShadowPixmap. This resource is only relevant if matteWidth is positive.

matteWidth (class MatteWidth)
: Specifies the width of the matte. The default is 0 (thus, no matte is used).

maximumClientSize (class MaximumClientSize)
: Specifies how a window is to be maximized, either to a specific size (*widthxheight*), or as much as possible in a certain direction (vertical or horizontal). If the value is of the form *widthxheight*, the width and height are interpreted in the units used by the client. For example, *xterm* measures width and height in font characters and lines.

  If maximumClientSize is not specified, and the WM_NORMAL_HINTS property is set, the default is obtained from it. If WM_NORMAL_HINTS is not set, the default is the size (including borders) that fills the screen.

  If maximumClientSize is not specified, *mwm* uses any value supplied to maximumMaximumSize.

useClientIcon (class UseClientIcon)
: If True, an icon image supplied by the client takes precedence over an icon image supplied by the user. The default is False.

usePPosition (class UsePPosition)
: Specifies whether *mwm* uses initial coordinates supplied by the client application. If True, *mwm* always uses the program specified position. If False, *mwm* never uses the program specified position. The default is nonzero, which means that *mwm* will use any program specified position except 0,0. Available as of *mwm* version 1.2.

windowMenu (class WindowMenu)
: Specifies a name for the Window Menu (which must be defined in the startup file). The default is DefaultWindowMenu. See the section ".mwmrc Startup File" earlier in this reference page and Chapter 13 of this guide.

Environment Variables

The following environment variables are used by *mwm*:

HOME The user's home directory.

LANG The language to be used for the *mwm* message catalog and the *mwm* startup file.

XBMLANGPATH
: Used to search for bitmap files.

XFILESEARCHPATH
: Used to determine the location of system-wide class resource files. If the LANG variable is set, the *$LANG* subdirectory is also searched.

XUSERFILESEARCHPATH, XAPPLRESDIR
: Used to determine the location of user-specific class resource files. If the LANG variable is set, the *$LANG* subdirectory is also searched.

MWMSHELL, SHELL
: MWMSHELL specifies the shell to use when executing a command supplied as an argument to the `f.exec` function. (See "mwm Functions" earlier in this reference page.) If MWMSHELL is not set, SHELL is used.

Files

/usr/lib/X11/$LANG/system.mwmrc
/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/Mwm
$HOME/$LANG/.mwmrc
$HOME/.mwmrc
$HOME/.motifbind—Used to install the virtual key bindings property on the root window.

See Also

X, Xserver, xdm, xrdb; Chapter 3, *Working in the X Environment*; Chapter 4, *More about the mwm Window Manager*; Chapter 13, *Customizing mwm.*

oclock

Name

oclock – display time of day in analog form.

Syntax

`oclock` [*options*]

Description

oclock displays the current time on an analog display. The clock face is smaller and more stylized than that for *xclock*. For example, there are no tick-marks, save for a "jewel" at the 12 o'clock position. The chief virtue of *oclock*, and the one thing that has made it popular, is that it makes use of the X shape extension, which supports non-rectangular windows. The default *oclock* window is round, but it can be resized into all kinds of interesting ovals.

Chapter 8, *Other Clients*, describes how to use the *oclock* client.

Options

oclock accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. (We've included some of the more commonly used Toolkit options later in this section.) In addition, *oclock* accepts the following application-specific options:

`-backing` *level*
: Specifies an appropriate level of backing store. *level* is one of `WhenMapped`, `Always`, or `NotUseful`.

`-hour` *color*
: Specifies a color for the hour hand of the clock.

`-jewel` *color*
: Specifies a color for the jewel of the clock.

`-minute` *color*
: Specifies a color for the minute hand of the clock.

`-noshape`
: Causes the clock not to use the shape extension for non-rectangular windows; in short, with `-noshape`, you get a square or rectangular clock. Note that the behavior is still different from *xclock -analog*. If you resize *xclock* so that its window is rectangular, the round clockface image is centered in the rectangle. With *oclock*, the border of the window is always the shape of the clock.

`-shape`
: Causes the clock to use the shape extension, which allows non-rectangular windows. You get the standard round clock face; this is the default.

`-transparent`
: Creates a transparent clock consisting only of the jewel, hands, and clock border. This creates an interesting visual effect: i.e., the background on which the *oclock* is placed can be seen through the clock.

The following standard X Toolkit options are commonly used with *oclock*:

`-bg` *`color`*
: Specifies a color for the background.

`-bw` *`pixels`*
: Specifies a width in pixels for the window border. As the Clock widget changes its border around quite a bit, this is most usefully set to zero.

`-fg` *`color`*
: Specifies a color for both the hands and the jewel of the clock.

Resources

You can specify the following nonstandard resources for *oclock*. Note that you can use either the clock widget or *oclock* in the resource specification. If you use the clock widget, you must precede it with a loose binding; if you use *oclock*, you must follow it with a loose binding. For example, these two resources would both produce an *oclock* with a red hour hand:

```
*clock.hour:  red
oclock*hour:  red
```

Since the latter resource is more specific, it has precedence. Thus, if a resource file contained both of the following:

```
*clock.hour:  red
oclock*hour:  blue
```

any subsequent instance of *oclock* would have a blue hour hand. See Chapter 11, *Setting Resources*, for more information.

Here are the resources you can set:

`backingStore` (class `BackingStore`)
: Specifies an appropriate level of backing store. Allowable values are `WhenMapped`, `Always`, or `NotUseful`.

`hour` (class `Foreground`)
: Specifies a color for the hour hand.

`jewel` (class `Foreground`)
: Specifies a color for the jewel of the clock.

`minute` (class `Foreground`)
: Specifies a color for the minute hand.

`shapeWindow` (class `ShapeWindow`)
: When false, causes the clock not to use the shape extension for non-rectangular windows. (See the `-noshape` option above.) The default is true (the shape extension is used and you get the default round clock face).

`transparent` (class `Transparent`)
If true, creates a transparent clock consisting only of the jewel, hands, and clock border. See also the `-transparent` option above. The default is false.

Colors

Although the default colors for the Clock widget are black and white, the widget was designed in color. Release 5 sees the addition of an application defaults file specifying colors for the various features of the *oclock*. (The Release 4 version of *oclock* requires you to specify every desired color in your own resources file.) If you would like your clock to be viewable in the prescribed colors, include the following resource definition in the #ifdef COLOR section of your *.Xresources* file (or whatever file you read with *xrdb*):

```
Clock*customization:    -color
```

On a color display, this will cause *oclock* to use the colors specified in the application defaults color customization file (generally */usr/lib/X11/app-defaults/Clock-color*). The default colors specified are:

```
Clock*Background: grey
Clock*BorderColor: light blue
Clock*hour: yellow
Clock*jewel: yellow
Clock*minute: yellow
```

Of course, you can opt to specify alternative colors in your own resource file.

For instructions on specifying resources, see Chapter 11 of this guide.

Files

/usr/lib/X11/app-defaults/Clock-color —Specifies color resources (as of Release 5).

See Also

X; Chapter 8, *Other Clients*; Chapter 11, *Setting Resources*; Volume Four, *X Toolkit Intrinsics Programming Manual*; Volume Five, *X Toolkit Intrinsics Reference Manual*.

Author

Keith Packard, MIT X Consortium.

resize

Reset Terminal for Window Size

Name

resize – utility to set TERMCAP and terminal settings to the current window size.

Syntax

resize [*options*]

Description

The *resize* client is provided for use with systems that lack the ability to automatically notify processes of window size changes. Normally, on operating systems that support terminal resizing, *xterm* sends a signal (e.g., SIGWINCH on BSD 4.3-derived UNIX systems) to notify processes running in the window that the window size has changed. These programs can adjust their behavior if necessary.

On systems that don't support terminal resizing, you can use the *resize* client. *resize* prints a shell command for setting the TERM and TERMCAP environment variables to indicate the current size of the *xterm* window from which the command is run. For this output to take effect, *resize* must either be evaluated as part of the command line (usually done with a shell alias or function) or else redirected to a file which can then be read in. From the C shell (usually known as */bin/csh*), the following alias could be defined in the user's *.cshrc*:

```
% alias rs 'set noglob; eval `resize`; unset noglob'
```

After resizing the window, the user would type:

```
% rs
```

Users of versions of the Bourne shell (usually known as */bin/sh*) that don't have command functions will need to send the output to a temporary file and then read it back in with the "." command:

```
$ resize >/tmp/out
$ . /tmp/out
```

See Chapter 5, *The xterm Terminal Emulator*, for more information.

Options

resize accepts the following options:

-u
: Indicates that Bourne shell commands should be generated even if the user's current shell isn't */bin/sh*.

-c
: Indicates that C shell commands should be generated even if the user's current shell isn't */bin/csh*.

-s [*rows columns*]
: Indicates that Sun console escape sequences will be used instead of the special *xterm* escape code. If *rows* and *columns* are given, *resize* will ask the *xterm* to resize itself. However, the window manager may choose to disallow the change.

The `-u` or `-c` must appear to the left of `-s` if both are specified.

Files

/etc/termcap
For the base termcap entry to modify.

~/.cshrc User's alias for the command.

See Also

csh(1), tset(1), xterm; Chapter 5, *The xterm Terminal Emulator*.

Bugs

There should be some global notion of display size; *termcap* and *terminfo* need to be rethought in the context of window systems. (Fixed in BSD 4.3 and Ultrix-32 1.2.)

Authors

Mark Vandevoorde, MIT Project Athena, and Edward Moy Berkeley.

sessreg

Manage uutmp/wtmp entries

Name

sessreg – manage utmp/wtmp entries for non-init clients.

Syntax

sessreg -a | -d [*options*] *user_name*

Description

sessreg is a simple program for managing utmp/wtmp entries for *xdm* sessions.

System V has a better interface to */etc/utmp* than BSD; it dynamically allocates entries in the file, instead of writing them at fixed positions indexed by position in */etc/ttys*.

To manage BSD-style *utmp* files, *sessreg* has two strategies. In conjunction with *xdm*, the -x option counts the number of lines in */etc/ttys* and then adds to that the number of the line in the *Xservers* file which specifies the display. The display name must be specified as the *line_name* using the -l option. This sum is used as the *slot_number* in */etc/utmp* that this entry will be written at. In the more general case, the -s option specifies the slot number directly. If for some strange reason your system uses a file other that */etc/ttys* to manage init, the -t option can direct *sessreg* to look elsewhere for a count of terminal sessions.

Conversely, System V managers will not ever need to use these options (-x, -s, and -t). To make the program easier to document and explain, *sessreg* accepts the BSD-specific flags in the System V environment and ignores them.

BSD also has a hostname field in the *utmp* file which doesn't exist in System V. This option is also ignored by the System V version of *sessreg*.

Options

-a Add this session to *utmp* or *wtmp*. Either -a or -d must be specified.

-d Delete this session from *utmp* or *wtmp*. Either -a or -d must be specified.

-h *host_name*
For BSD hosts, this is set to indicate that the session was initiated from a remote host. In typical *xdm* usage, this option is not used.

-l *line_name*
Describes the "line" name of the entry. For terminal sessions, this is the final path-name segment of the terminal device filename (for example, ttyd0). For X sessions, it should probably be the local display name given to the users session (for example, :0). If none is specified, the terminal name will be determined with *ttyname*(3) and stripped of leading components.

-s *slot_number*
Each potential session has a unique slot number in BSD systems; most are identified by the position of the *line_name* in the */etc/ttys* file. This option overrides the default position determined with *ttyslot*(3). It is inappropriate for use with *xdm*; the -x option is more useful.

`-t` *`ttys_file`*
: Specifies an alternate file which the `-x` option will use to count the number of terminal sessions on a host.

`-u` *`utmp_file`*
: Specifies an alternate *utmp* file, instead of */etc/utmp*. The special name `none` disables writing records to */etc/utmp*.

`-w` *`wtmp_file`*
: Specifies an alternate *wtmp* file, instead of */usr/adm/wtmp* for BSD or */etc/wtmp* for System V. The special name `none` disables writing records to */usr/adm/wtmp*.

`-x` *`Xservers_file`*
: As X sessions are one-per-display, and each display is entered in this file, this options sets the *`slot_number`* to be the number of lines in the *`ttys_file`* plus the index into this file that the *`line_name`* is found.

Usage

In *Xstartup*, place a call like:

```
sessreg -a -l $DISPLAY -x /usr/lib/X11/xdm/Xservers $USER
```

and in *Xreset*:

```
sessreg -d -l $DISPLAY -x /usr/lib/X11/xdm/Xservers $USER
```

See Also

xdm.

Author

Keith Packard, MIT X Consortium.

showfont

Display Font Data under Font Server

Name

showfont – font dumper for the X font server.

Syntax

showfont -server *server_name*[*options*] -fn *pattern*

Description

showfont displays data about the font that matches the given *pattern*. This client is new in Release 5 and is intended to be run with the font server (*fs*).

The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

showfont displays the contents of font files in the Portable Compiled Font format produced by *bdftopcf*. The information displayed includes the value of each of the font properties. (For more information, see Appendix M, *X Logical Font Description Conventions*, in Volume Zero, *X Protocol Reference Manual*. See also information on font metrics in Section 6.2.3, "Character Metrics," of Volume One, *Xlib Programming Manual*.)

Options

-bitmap_pad *padding_unit*
: Specifies the bitmap padding unit of the font. Acceptable values are 0, 1, or 2, where 0 is ImageRectMin, 1 is ImageRectMaxWidth and 2 is ImageRectMaxWidth.

-end *character_number*
: Specifies that the range of the characters displayed should end with *character_number* (a decimal number).

-extents_only
: Indicates that only the font's extents should be displayed.

-l
: Indicates that the bit order of the font is least significant bit first.

-L
: Indicates that the byte order of the font is least significant byte first.

-m
: Indicates that the bit order of the font is most significant bit first.

-M
: Indicates that the byte order of the font is most significant byte first.

-pad *scanpad_unit*
: Specifies the scanpad unit of the font. Acceptable values are: 1, 2, 4, or 8.

-server *server_name*
: Specifies a particular font server. The *server_name* generally has the form *transport/host:port*. If the FONTSERVER environment variable is not defined, this option must be given.

`-start` *`character_number`*
Indicates the range of the characters to display should start with *`character_number`* (a decimal number).

`-unit` *`scanline_unit`*
Specifies the scanline unit of the font. Acceptable values are: `1`, `2`, `4`, or `8`.

See Also

fs, showsnf, xlsfonts; Chapter 6, *Font Specification*; Chapter 7, *Graphics Utilities*; Appendix M, *X Logical Font Description Conventions*, in Volume Zero, *X Protocol Reference Manual*; Section 6.2.3, "Character Metrics," in Volume One, *Xlib Programming Manual*.

Environment Variables

FONTSERVER
To get the default font server.

Copyright

Author

Dave Lemke, Network Computing Devices, Inc.

showrgb

Display RGB Color Database

Name

showrgb – uncompile an RGB color name database.

Syntax

showrgb [*database*]

Description

The *showrgb* program reads an RGB color name database compiled for use with the *dbm* database routines and converts it back to source form, printing the result to standard output. The default database is the one that X was built with, and may be overridden on the command line. Specify the database name without the *.pag* or *.dir* suffix.

Files

/usr/lib/X11/rgb.txt
: Text version of the default database.

/usr/lib/X11/rgb.dir

/usr/lib/X11/rgb.pag
: Machine-readable database files.

See Also

The discussion of color in Chapter 12, *Specifying Color*.

Name

showsnf – print contents of an SNF file to standard output.

Syntax

showsnf [*options*] *snf_file*

Description

This client has been removed from the standard distribution in Release 5. If you are running the font server (*fs*), use the *showfont* client instead.

showsnf displays the contents of font files in the Server Natural Format produced by *bdftosnf*. The information displayed includes the value of each of the font properties (see Appendix M, *X Logical Font Description Conventions, Release 5*, in Volume Zero, *X Protocol Reference Manual*), as well as information on font metrics (see Section 6.2.3, *Character Metrics*, in Volume One, *Xlib Programming Manual*).

showsnf is usually used only to verify that a font file hasn't been corrupted or to convert the individual glyphs into arrays of characters for proofreading or for conversion to some other format.

Options

showsnf accepts the following options:

`-g` Indicates that character glyph bitmaps should be printed.

`-l` Indicates that the bit order of the font is least significant bit first.

`-L` Indicates that the byte order of the font is least significant byte first.

`-m` Indicates that the bit order of the font is most significant bit first.

`-M` Indicates that the byte order of the font is most significant byte first.

`-p`*number*
Specifies the glyph padding of the font.

`-u`*number*
Specifies the scanline unit of the font.

`-v` Indicates that bearings and sizes should be printed for each character in the font. (These are in an ASCII format similar to that produced by *bmtoa*. They can be converted to standard X bitmaps using *atobm*, and then edited with *bitmap*.)

See Also

X, Xserver, bdftosnf, showfont, bitmap; Chapter 6, *Font Specification*.

Bugs

There is no way to print out only a single glyph.

viewres

Name

viewres – Athena widget class browser.

Syntax

`viewres` [*option*]

Description

The *viewres* program (available as of Release 5) displays a tree showing the widget class hierarchy of the Athena Widget Set. Each node in the tree can be expanded to show the resources that the corresponding class adds (i.e., does not inherit from its parent) when a widget is created. This application allows the user to view the structure and inherited resources for the Athena Widget Set.

Options

viewres accepts all of the standard X Toolkit command-line options, as well as the following additional options:

`-top` *name*
: Specifies the name of the highest widget in the hierarchy to display. This is typically used to limit the display to a subset of the tree. The default is `Object`.

`-variable`
: Indicates that the widget variable names (as declared in header files) should be displayed in the nodes rather than the widget class name. This is sometimes useful to distinguish widget classes that share the same name (such as `Text`).

`-vertical`
: Indicates that the tree should be displayed top to bottom rather than left to right. See also the Layout Vertical item in the "View Menu" section following.

View Menu

The way in which the tree is displayed may be changed through the entries in the View menu:

Layout Horizontal
: Causes the tree to be laid out from left to right. This operation may also be performed with the `SetOrientation(West)` translation.

Layout Vertical
: Causes the tree to be laid out from top to bottom. This operation may also be performed with the `SetOrientation(North)` translation. (To specify a vertical layout: on the command line, run *viewres* with the `-vertical` option; or in a resource file, use `*Tree.Gravity: north`.)

Show Variable Names
: Causes the node labels to be set to the variable names used to declare the corresponding widget class. This operation may also be performed with the `SetLabelType(`*variable*`)` translation.

Show Class Names

Causes the node labels to be set to the class names used when specifying resources. This operation may also be performed with the `SetLabelType(class)` translation.

Show Resource Boxes

Expands the selected nodes (see next section) to show the new widget and constraint resources. This operation may also be performed with the `Resources(on)` translation.

Hide Resource Boxes

Removes the resource displays from the selected nodes (usually to conserve space). This operation may also be performed with the `Resources(off)` translation.

Select Menu

Resources for a single widget class can be displayed by clicking the second pointer button **Button2** on the corresponding node, or by adding the node to the selection list with **Button1** and using the Show Resource Boxes entry in the View menu. Since **Button1** actually toggles the selection state of a node, clicking on a selected node will cause it to be removed from the selected list.

Collections of nodes may also be selected through the various entries in the Select menu:

Unselect All

Removes all nodes from the selection list. This operation may also be performed with the `Select(nothing)` translation.

Select All

Adds all nodes to the selection list. This operation may also be performed with the `Select(all)` translation.

Invert All

Adds unselected nodes to, and removes selected nodes from, the selection list. This operation may also be performed with the `Select(invert)` translation.

Select Parent

Selects the immediate parents of all selected nodes. This operation may also be performed with the `Select(parent)` translation.

Select Ancestors

Recursively selects all parents of all selected nodes. This operation may also be performed with the `Select(ancestors)` translation.

Select Children

Selects the immediate children of all selected nodes. This operation may also be performed with the `Select(children)` translation.

Select Descendants

Recursively selects all children of all selected nodes. This operation may also be performed with the `Select(descendants)` translation.

Select Has Resources
: Selects all nodes that add new resources (regular or constraint) to their corresponding widget classes. This operation may also be performed with the `Select(resources)` translation.

Select Shown Resource Boxes
: Selects all nodes whose resource boxes are currently expanded (usually so that they can be closed with the Hide Resource Boxes) menu option. This operation may also be performed with the `Select(shown)` translation.

Resources

viewres defines the following application-specific resources:

`showVariable` (class `showVariable`)
: If true, indicates that the widget variable names (as declared in header files) should be displayed in the nodes rather than the widget class names. This is sometimes useful to distinguish widget classes that share the same name (such as `Text`). The default is false.

`topObject` (class `TopObject`)
: Specifies the name of the highest widget in the hierarchy to display. This is typically used to limit the display to a subset of the tree. The default is `Object`.

Note that you can specify the direction the tree goes using the `Tree` widget and `Gravity` resource. For a vertical (top to bottom) orientation, use:

```
*Tree.Gravity: north
```

The default orientation is `west` (horizontal, read left to right). You can also specify `south` for a bottom to top tree, or `east` for right to left, but neither of these orientations is particularly intuitive in English.

Actions

The following application actions are provided:

`Quit()`
: Causes *viewres* to exit.

`Resources(`*option*`)`
: Turns `on`, `off`, or `toggles` the resource boxes for the selected nodes. If invoked from within one of the nodes (through the keyboard or pointer), only that node is used.

`SetLabelType(`*type*`)`
: Sets the node labels to display the widget variable or class names, according to the argument *type*.

`SetOrientation(`*direction*`)`
: Sets the root of the tree to be one of the following areas of the window: `West`, `North`, `East`, or `South`.

`Select(`*`what`*`)`

Selects the indicated nodes, as described in the View Menu section: `nothing` (unselects all nodes), `invert`, `parent`, `ancestors`, `children`, `descendants`, `resources`, `shown`.

Widgets

Resources may be specified for the following widgets:

```
Viewres viewres
        Paned pane
                Box buttonbox
                        Command quit
                        MenuButton view
                                SimpleMenu viewMenu
                                        SmeBSB layoutHorizontal
                                        SmeBSB layoutVertical
                                        SmeLine line1
                                        SmeBSB namesVariable
                                        SmeBSB namesClass
                                        SmeLine line2
                                        SmeBSB viewResources
                                        SmeBSB viewNoResources
                        MenuButton select
                                SimpleMenu selectMenu
                                        SmeBSB unselect
                                        SmeBSB selectAll
                                        SmeBSB selectInvert
                                        SmeLine line1
                                        SmeBSB selectParent
                                        SmeBSB selectAncestors
                                        SmeBSB selectChildren
                                        SmeBSB selectDescendants
                                        SmeLine line2
                                        SmeBSB selectHasResources
                                        SmeBSB selectShownResources
                Form treeform
                        Porthole porthole
                                Tree tree
                                        Box variable_name
                                                Toggle variable_name
                                                List variable_name
                        Panner panner
```

where *`variable_name`* is the widget variable name of each node.

Files

/usr/lib/X11/app-defaults/Viewres
Specifies required resources.

See Also

X, appres, editres, listres, xrdb; Chapter 11, *Setting Resources*; Volume Four, *X Toolkit Intrinsics Programming Manual*; Volume Five, *X Toolkit Intrinsics Reference Manual*.

Author

Jim Fulton, MIT X Consortium.

xauth

Name

xauth – X authority file utility.

Syntax

xauth [*options*] [*command arguments*]

Description

The *xauth* program is used to edit and display authorization information used when connecting to the X server. *xdm* can be configured to generate this authorization information when a user logs on. *xauth* is then used to extract authorization records from one machine and merge them in on another (as is the case when using remote logins or to grant access to other users). Note that this program does *not* contact the X server.

For an introduction to user-based access control and the *xauth* program, see Appendix A, *Managing Your Environment*. Volume Eight, *X Window System Administrator's Guide*, provides a more detailed discussion of these and other security-related topics.

Options

The following options may be used with *xauth*. They may be given individually (for example, `-q -i`) or may be combined (for example, `-qi`).

`-b` Indicates that *xauth* should attempt to break any authority file locks before proceeding and should only be used to clean up stale locks.

`-f` *authfile*
Specifies the name of the authority file to use. By default, *xauth* will use the file specified by the XAUTHORITY environment variable or *.Xauthority* in the user's home directory.

`-i` Indicates that *xauth* should ignore any authority file locks. Normally, *xauth* will refuse to read or edit any authority files that have been locked by other programs (usually *xdm* or another *xauth*).

`-q` Indicates that *xauth* should operate quietly and not print unsolicited status messages. This is the default if an *xauth* command is given on the command line or if the standard output is not directed to a terminal.

`-v` Indicates that *xauth* should operate verbosely and print status messages indicating the results of various operations (for example, how many records have been read in or written out). This is the default if *xauth* is reading commands from its standard input and its standard output is directed to a terminal.

Commands

Commands may be entered interactively, on the *xauth* command line, or in scripts. The following commands may be used to manipulate authority files (or obtain information):

`?` A short list of the valid commands is printed on the standard output.

`add` *displayname protocolname hexkey*

An authorization entry for the indicated *displayname* using the given *protocolname* and *hexkey* data is added to the authorization file.

At present, three *protocolname*s are supported in the standard X distribution: MIT-MAGIC-COOKIE-1, XDM-AUTHORIZATION-1, and SUN-DES-1. Note that *xauth* will not give you an error if you specify an invalid protocol name. A protocol name consisting of just a single period is treated as an abbreviation for MIT-MAGIC-COOKIE-1. (See Volume Eight, *X Window System Administrator's Guide*, for more information.)

The *hexkey* data is specified as an even-lengthed string of hexadecimal digits, each pair representing one octet. The first digit of each pair gives the most significant 4 bits of the octet and the second digit of the pair gives the least significant 4 bits. For example, a 32 character hexkey would represent a 128-bit value.

`exit` If any modifications have been made, the authority file is written out (if allowed), and the program exits. An end-of-file is treated as an implicit exit command.

`help` [*string*]

A description of all commands that begin with the given *string* (or all commands, if no string is given) is printed on the standard output.

`info` Information describing the authorization file, whether or not any changes have been made, and from where *xauth* commands are being read is printed on the standard output.

[n]`extract` *filename displayname . . .*

Authorization entries for each of the specified displays are written to the indicated file. If the `nextract` command is used, the entries are written in a numeric format suitable for non-binary transmission (such as secure electronic mail). The extracted entries can be read back in using the `merge` and `nmerge` commands. If the filename consists of just a single dash, the entries will be written to the standard output.

[n]`list` [*displayname...*]

Authorization entries for each of the specified displays (or all, if no displays are named) are printed on the standard output. If the `nlist` command is used, entries will be shown in the numeric format used by the `nextract` command; otherwise, they are shown in a textual format. Key data is always displayed in the hexadecimal format given in the description of the `add` command.

[n]`merge` [*filename...*]

Authorization entries are read from the specified files and are merged into the authorization database, superceding any matching existing entries. If the `nmerge` command is used, the numeric format given in the description of the `extract` command is used. If a filename consists of just a single dash, the standard input will be read if it hasn't been read before.

`quit` The program exits, ignoring any modifications. This may also be accomplished by pressing the interrupt character.

`remove` *`displayname...`*
Authorization entries matching the specified displays are removed from the authority file.

`source` *`filename`*
The specified *`filename`* is treated as a script containing *xauth* commands to execute. In such a file, blank lines and lines beginning with a sharp sign (#) are ignored. A single dash may be used to indicate the standard input, if it hasn't already been read.

Display Names

Display names for the `add`, `[n]extract`, `[n]list`, `[n]merge`, and `remove` commands use the same format as the DISPLAY environment variable and the common `-display` command-line option. Display-specific information (such as the screen number) is unnecessary and will be ignored. Same-machine connections (such as local-host sockets, shared memory, and the Internet Protocol hostname *`localhost`*) are referred to as *`hostname`*`/unix:`*`displaynumber`* so that local entries for different machines may be stored in one authority file.

Example

The most common use for *xauth* is to extract the entry for the current display, copy it to another machine, and merge it into the user's authority file on the remote machine:

```
% xauth extract - $DISPLAY | rsh other xauth merge -
```

Environment Variables

This *xauth* program uses the following environment variables:

XAUTHORITY
To get the name of the authority file to use if the `-f` option isn't used. If this variable is not set, *xauth* will use *.Xauthority* in the user's home directory.

HOME To get the user's home directory if XAUTHORITY isn't defined.

Bugs

Users that have unsecure networks should take care to use encrypted file transfer mechanisms to copy authorization entries between machines. Similarly, the MIT-MAGIC-COOKIE-1 protocol is not very useful in unsecure environments. Sites that are interested in additional security may need to use encrypted authorization mechanisms such as Kerberos.

Spaces are currently not allowed in the protocol name. Quoting could be added for the truly perverse.

See Also

X, Xserver, xdm; Appendix A, *Managing Your Environment*; Volume Eight, *X Window System Administrator's Guide*.

Author

Jim Fulton, MIT X Consortium.

xbiff

Name

xbiff – mail notification program for X.

Syntax

xbiff [*options*]

Description

The *xbiff* program displays a little image of a mailbox. When there is no mail in the user's mailbox, the flag on the mailbox is down. When mail arrives, the flag goes up and the mailbox beeps. By default, pressing any pointer button in the image forces *xbiff* to remember the current size of the mail file as being the "empty" size and to lower the flag.

This program is nothing more than a wrapper around the Athena Mailbox widget.

The default mailbox is 48 pixels on each side and is centered in the window. If you are using *mwm* without customization, the size of the *xbiff* image will be slightly largely to allow for the frame. To suppress part or all of this window decoration, see Chapter 13, *Customizing mwm*, and the *mwm* reference page.

See Chapter 8, *Other Clients*, for instructions on using *xbiff*.

Options

xbiff accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. (We've included some of the more commonly used Toolkit options later in this section.) In addition, *xbiff* accepts the following application-specific options:

`-file` *filename*
: Specifies the name of the file that should be monitored. By default, *xbiff* watches */usr/spool/mail/**username***, where ***username*** is your login name.

`-help` Indicates that a brief summary of the allowed options should be printed on the standard error.

`-shape`
: Indicates that the mailbox window should be shaped if masks for the empty or full images are given.

`-update` *seconds*
: Specifies the frequency in seconds at which *xbiff* should update its display. If the mailbox is obscured and then exposed, it will be updated immediately. The default is 30 seconds.

`-volume` *percentage*
: Specifies how loud the bell should be rung when new mail comes in.

Resources

xbiff is implemented using a simple widget, the Mailbox widget from the Athena Widget Set. The application class name is `XBiff`. *xbiff* understands all of the Core resource names and classes as well as those from the Mailbox widget. The resources you might want to set are listed below:

`checkCommand` (class `CheckCommand`)
: Specifies a shell command to be executed to check for new mail rather than examining the size of `file`. The specified string value is used as the argument to a *system*(3) call and may therefore contain I/O redirection. An exit status of 0 indicates that new mail is waiting; 1 indicates that there has been no change in size; and 2 indicates that the mail has been cleared. By default, no shell command is provided.

`emptyPixmap` (class `Pixmap`)
: Specifies a bitmap to be shown when no new mail is present. The default is `flagdown`.

`emptyPixmapMask` (class `PixmapMask`)
: Specifies a mask for the bitmap to be shown when no new mail is present. The default is none.

`file` (class `File`)
: Specifies the name of the file to monitor. The default is to watch */usr/spool/mail/**username***, where ***username*** is your login name.

`flip` (class `Flip`)
: Specifies whether or not the image that is shown when mail has arrived should be inverted. The default is true.

`foreground` (class `Foreground`)
: Specifies the color for the foreground.

`fullPixmap` (class `Pixmap`)
: Specifies a bitmap to be shown when new mail has arrived. The default is `flagup`.

`fullPixmapMask` (class `PixmapMask`)
: Specifies a mask for the bitmap to be shown when new mail has arrived. The default is none.

`height` (class `Height`)
: Specifies the height of the mailbox. The default is 48 pixels.

`onceOnly` (class `Boolean`)
: Specifies that the bell is rung only the first time new mail is found and is not rung again until at least one interval has passed with no mail waiting. The window will continue to indicate the presence of new mail until it has been retrieved. The default is false.

`reverseVideo` (class `ReverseVideo`)
: Specifies that the foreground and background should be reversed.

`shapeWindow` (class `ShapeWindow`)
Specifies whether or not the mailbox window should be shaped to the given `full-PixmapMask` and `emptyPixmapMask`. The default is false.

`update` (class `Interval`)
Specifies the frequency in seconds at which the mail should be checked. The default is 30.

`volume` (class `Volume`)
Specifies how loud the bell should be rung. The default is 33 percent.

`width` (class `Width`)
Specifies the width of the mailbox. The default is 48 pixels.

Widget Hierarchy

xbiff is implemented using a single widget, the Athena Mailbox widget. All applicable resources of the widget are listed above. The class and instance hierarchy is shown below:

```
Xbiff   xbiff
    Mailbox  mailbox
```

See Appendix G, *Widget Resources*, for a list of resources that can be set for the Athena widgets.

Actions

The Mailbox widget provides the following actions for use in event translations:

`check()`
Causes the widget to check for new mail and display the flag appropriately.

`unset()`
Causes the widget to lower the flag until new mail comes in.

`set()` Causes the widget to raise the flag until the user resets it.

The default translation is:

```
<ButtonPress>:unset()
```

See Also

X, xrdb, stat(2); Chapter 8, *Other Clients*; Appendix G, *Widget Resources*.

Author

Jim Fulton, MIT X Consortium;
Additional hacks by Ralph Swick, DEC/MIT Project Athena.

xcalc

X-Based Scientific Calculator

Name

xcalc – scientific calculator for X.

Syntax

`xcalc` [*options*]

Description

xcalc is a scientific calculator desktop accessory that can emulate a TI-30 or an HP-10C. The number in the calculator display can be selected, allowing you to paste the result of a calculation into text. See Chapter 8, *Other Clients*, for instructions on using the calculator.

Since Release 4, the *xcalc* buttons have been an oval shape (they are rectangular in earlier versions) and the window is somewhat smaller overall than it was in previous releases. For those of you who like the size of the calculator under R3, try a geometry specification of approximately 167 × 222 and specify rectangular buttons using the resource setting:

```
xcalc*shapeStyle:  rectangular
```

`shapeStyle` is a resource of the Athena `Command` widget. See "Widget Hierarchy" later in this reference page for a diagram of *xcalc*'s structure and Appendix G for a list of the resources you can set for various widgets.

Options

xcalc accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. In addition, *xcalc* accepts the following application-specific options:

`-rpn` Indicates that Reverse Polish Notation should be used. In this mode, the calculator will look and behave like an HP-10C. Without this flag, it will emulate a TI-30.

`-stip`, `-stipple`
Indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays, this improves the appearance.

Calculator Operations

Pointer Usage

Operations may be performed with pointer button 1 (usually the leftmost button), or in many cases, with the keyboard. Many common calculator operations have keyboard equivalents, which are called accelerators, because they facilitate data entry. There are several ways to cause *xcalc* to exit: pressing the AC key of the TI calculator or the ON key of the HP calculator with pointer button 3 (usually the rightmost button), and typing q, Q, or CTRL-C while the pointer is in the *xcalc* window.

Calculator Key Usage (TI Mode)

The number keys, the +/– key, and the +, –, *, /, and = keys all do exactly what you would expect them to. It should be noted that the operators obey the standard rules of precedence.

Thus, entering "3+4*5=" results in 23, not 35. Parentheses can be used to override this. For example, "(1+2+3)*(4+5+6)=" is evaluated as "6*15=" which results in 90.

The action associated with each function is given below. These are useful if you are interested in defining a custom calculator. The action used for all digit keys is `digit(`*n*`)`, where *n* is the corresponding digit, 0-9.

The keys are described below:

1/x Replaces the number in the display with its reciprocal. The corresponding action is `reciprocal()`.

x^2 Squares the number in the display. The corresponding action is `square()`.

SQRT Evaluates the square root of the number in the display. The corresponding action is `squareRoot()`.

CE/C When pressed once, clears the number in the display without clearing the state of the machine. Allows you to re-enter a number if you make a mistake. Pressing it twice clears the state also. The corresponding action is `clear()`.

AC Clears everything: the display, the state, and the memory. Pressing it with the third (usually the right) button "turns off" the calculator, in that it exits the program. The corresponding action to clear the state is `off()`; to quit, the action is `quit()`.

INV Inverts the meaning of the function keys. See the individual function keys for details. The corresponding action is `inverse()`.

sin Computes the sine of the number in the display, as interpreted by the current DRG mode (see DRG, below). If inverted, it computes the arcsine. The corresponding action is `sine()`.

cos Computes the cosine, or arccosine when inverted. The corresponding action is `cosine()`.

tan Computes the tangent, or arctangent when inverted. The corresponding action is `tangent()`.

DRG Changes the DRG mode, as indicated by DEG, RAD, or GRAD at the bottom of the calculator "liquid crystal" display. When in DEG mode, numbers in the display are taken as being degrees. In RAD mode, numbers are in radians, and in GRAD mode, numbers are in gradians. When inverted, the DRG key has the handy feature of converting degrees to radians to gradians and vice versa. For example, put the calculator into DEG mode and type "45 INV DRG". The calculator should display approximately .785398, which is 45 degrees converted to radians. The corresponding action is `degree()`.

e The constant "e" (2.7182818 . . .). The corresponding action is `e()`.

EE — Used for entering exponential numbers. For example, to enter "`-2.3E-4`" you would type "`2 . 3 +/- EE 4 +/-`". The corresponding action is `scientific()`.

log — Calculates the log (base 10) of the number in the display. When inverted, it raises 10.0 to the number in the display. For example, entering "`3 INV log`" should result in 1000. The corresponding action is `logarithm()`.

ln — Calculates the log (base e) of the number in the display. When inverted, it raises "e" to the number in the display. For example, entering "`e ln`" should result in 1. The corresponding action is `naturalLog()`.

y^x — Raises the number on the left to the power of the number on the right. For example, "`2 y^x 3 =`" results in 8, which is 2^3. Also, "`(1+2+3) y^x (1+2)=`" is evaluated as "`6 y^x 3=`" which results in 216. The corresponding action is `power()`.

PI — The constant "pi". (3.1415927 . . . .) The corresponding action is `pi()`.

x! — Computes the factorial of the number in the display. The number in the display must be an integer in the range 0-500, though depending on your math library, it might overflow long before that. The corresponding action is `factorial()`.

(— Left parenthesis. The corresponding action for TI calculators is `leftParen()`.

) — Right parenthesis. The corresponding action for TI calculators is `rightParen()`.

/ — Division. The corresponding action is `divide()`.

* — Multiplication. The corresponding action is `multiply()`.

– — Subtraction. The corresponding action is `subtract()`.

+ — Addition. The corresponding action is `add()`.

= — Perform calculation. The TI-specific action is `equal()`.

STO — Copies the number in the display to the memory location. The corresponding action is `store()`.

RCL — Copies the number from the memory location to the display. The corresponding action is `recall()`.

SUM — Adds the number in the display to the number in the memory location. The corresponding action is `sum()`.

EXC — Swaps the number in the display with the number in the memory location. The corresponding action is `exchange()`.

+/– Negate (change sign). The corresponding action is `negate()`.

. Decimal point. The corresponding action is `decimal()`.

Calculator Key Usage (RPN mode)

The number keys, CHS (change sign), +, –, *, /, and ENTR keys all do exactly what you would expect them to. Many of the remaining keys are the same as in TI (default) mode. The differences are detailed below. The action for the ENTR key is `enter()`.

<– Backspace key that can be used while entering a number. It will erase digits from the display. (See "Bugs.") Inverse backspace clears the X register. The corresponding action is `back()`.

ON Clears everything: the display, the state, and the memory. Pressing it with the third (usually the right) pointer button "turns off" the calculator, in that it exits the program. The corresponding action to clear the state is `off()`; to quit, the action is `quit()`.

INV Inverts the meaning of the function keys. This would be the "f" key on an HP calculator, but *xcalc* does not display multiple legends on each key. See the individual function keys for details.

10^x Raises 10.0 to the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display. The corresponding action is `tenpower()`.

e^x Raises "e" to the number in the top of the stack. When inverted, it calculates the log (base e) of the number in the display. The corresponding action is `epower()`.

STO Copies the number in the top of the stack to one of ten memory locations. The desired memory is specified by pressing this key and then pressing a digit key.

RCL Pushes the number from the specified memory location onto the stack.

SUM Adds the number on top of the stack to the number in the specified memory location.

x:y Exchanges the numbers in the top two stack positions, the X and Y registers. The corresponding action is `XexchangeY()`.

R v Rolls the stack downward. When inverted, it rolls the stack upward. The corresponding action is `roll()`.

Blank keys were used for programming functions on the HP-10C. Their functionality has not been duplicated in *xcalc*.

Keyboard Equivalents (Accelerators)

If you have the pointer in the *xcalc* window, you can use the keyboard to enter numbers and other keys. Almost all of the calculator keys have keyboard equivalents, which are known as *accelerators* because they speed entry. The number keys, the operator keys, and the parentheses all have the obvious equivalents. The accelerators defined by *xcalc* are listed in the following table:

TI Key	HP Key	Keyboard Accelerator	TI Function	HP Function
SQRT	SQRT	r	`squareRoot()`	`squareRoot()`
AC	ON	space	`clear()`	`clear()`
AC	<–	Delete	`clear()`	`back()`
AC	<–	Backspace	`clear()`	`back()`
AC	<–	Control-H	`clear()`	`back()`
AC		Clear	`clear()`	
AC	ON	q	`quit()`	`quit()`
AC	ON	Control-C	`quit()`	`quit()`
INV	i	i	`inverse()`	`inverse()`
sin	s	s	`sine()`	`sine()`
cos	c	c	`cosine()`	`cosine()`
tan	t	t	`tangent()`	`tangent()`
DRG	DRG	d	`degree()`	`degree()`
e		e	`e()`	
ln	ln	l	`naturalLog()`	`naturalLog()`
y^x	y^x	^	`power()`	`power()`
PI	PI	p	`pi()`	`pi()`
x!	x!	!	`factorial()`	`factorial()`
(		(	`leftParen()`	
)		)	`rightParen()`	
/	/	/	`divide()`	`divide()`
*	*	*	`multiply()`	`multiply()`
–	–	–	`subtract()`	`subtract()`
+	+	+	`add()`	`add()`
=		=	`equal()`	
0...9	0...9	0...9	`digit()`	`digit()`
.	.	.	`decimal()`	`decimal()`
+/-	CHS	n	`negate()`	`negate()`
	x:y	x		`XexchangeY()`
	ENTR	Return		`enter()`
	ENTR	Linefeed		`enter()`

Resources

The application class name is `XCalc`.

xcalc defines the following application resources:

`cursor` (class `Cursor`)
: The name of the symbol used to represent the pointer. The default is hand2. See Appendix D for a list of cursor names.

`rpn` (class `Rpn`)
: A Boolean value that specifies whether or not the rpn mode should be used. The default is off—that is, the calculator will be used in TI mode.

`stipple` (class `Stipple`)
: A Boolean value that indicates whether or not the background should be stippled. The default is on for monochrome displays, and off for color displays.

Widget Hierarchy

In addition, you can specify resources for each of the widgets that make up *xcalc*. In the notation below, indentation indicates the hierarchical structure of the widgets. The widget class name is given first, followed by the widget instance name.

```
XCalc xcalc
            Form  ti  or  hp          (The name depends on the mode)
                Form  bevel
                    Form  screen
                        Label  M      (The memory indicator on the screen)
                        Toggle LCD    (Where the data is displayed)
                        Label  INV    (The inverted indicator on the display)
                        Label  DEG    (The degrees indicator on the display)
                        Label  RAD    (The radians indicator on the display)
                        Label  GRAD   (The gradians indicator on the display)
                        Label  P      (The Parenthesis indicator on the display)
                Command  button1      (The actual calculator buttons)
                Command  button2      (Buttons are numbered from right to left)
                Command  button3      (See the app-defaults file for associations
                                       and so on...)
                Command  button38     (Between widget names and default labels)
                Command  button39
                Command  button40     (Only 39 buttons in HP mode)
```

See Appendix G, *Widget Resources*, for a list of resources that can be set for the Athena widgets.

Customization

xcalc has an enormous application defaults file, which specifies the position, label, and function of each key on the calculator. It also gives translations to serve as keyboard accelerators. Because these resources are not specified in the source code, you can create a customized calculator by writing a private application defaults file, using the Athena Command and Form

widget resources to specify the size and position of buttons, the label for each button, and the function of each button.

The foreground and background colors of each calculator key can be individually specified. For the TI calculator, a classical color resource specification might be:

```
XCalc.ti.Command.background:        grey50
XCalc.ti.Command.foreground:        white
```

For each of buttons 20, 25, 30, 35, and 40, specify:

```
XCalc.ti.button20.background:       black
XCalc.ti.button20.foreground:       white
```

For each of buttons 22, 23, 24, 27, 28, 29, 32, 33, 34, 37, 38, and 39:

```
XCalc.ti.button22.background:       white
XCalc.ti.button22.foreground:       black
```

Colors

As of Release 5, you can opt to use a predefined set of colors for the TI calculator. These colors are provided in a second application defaults file. If you would like your calculator to be viewable in the standard colors, include the following resource definition in your *.Xresources* file (or whatever file you read with *xrdb*):

```
*customization:   -color
```

This will cause *xcalc* to pick up the colors in the app-defaults color customization file (generally */usr/lib/X11/app-defaults/XCalc-color*).

Files

/usr/lib/X11/app-defaults/XCalc
: Specifies required resources.

/usr/lib/X11/app-defaults/XCalc-color
: Specifies color resources for the TI calculator (as of Release 5).

See Also

X, xrdb; Chapter 8, *Other Clients*; Appendix G, *Widget Resources*.

Bugs

In HP mode, a bug report claims that the sequence of keys 5, ENTR, and <– should clear the display, but it doesn't.

Authors

John Bradley, University of Pennsylvania;
Mark Rosenstein, MIT Project Athena.

xclipboard

Name

xclipboard – X clipboard client.

Syntax

xclipboard [*options*]

Description

The *xclipboard* program is used to collect and display text selections that are sent to the CLIPBOARD by other clients. It is typically used to save CLIPBOARD selections for later use. Chapter 5, *The xterm Terminal Emulator*, describes how to use the *xclipboard* client. See Chapter 11, *Setting Resources*, for instructions on customizing *xterm* to send selections to the CLIPBOARD.

Since *xclipboard* uses a Text widget to display the contents of the clipboard, text sent to the CLIPBOARD may be reselected for use in other applications. The contents may also be edited, using any of the editing commands built into the Text widget. (See the reference page for *xedit* for details.)

xclipboard stores each CLIPBOARD selection as a separate string, each of which can be selected. Each time CLIPBOARD is asserted by another application, *xclipboard* transfers the contents of that selection to a new buffer and displays it in the text window. Buffers are never automatically deleted, so you'll want to use the delete button to get rid of useless items.

xclipboard also responds to requests for the CLIPBOARD selection from other clients by sending the entire contents of the currently displayed buffer.

An *xclipboard* window has the following buttons across the top:

Quit	Exits *xclipboard*.
Delete	Deletes current buffer and displays the next one.
New	Creates a new buffer with no contents. Useful in constructing a new CLIPBOARD selection by hand.
Save	Saves the currently displayed selection to a file. Available as of Release 5.
Next	Displays the next buffer in the list.
Previous	Displays the previous buffer.

To the right of these command buttons is a small box displaying a number corresponding to the selection being displayed. The box has been added in Release 5.

Options

xclipboard accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. In addition, *xclipboard* accepts the following application-specific options:

-nw	Indicates that long lines of text should not wrap around. This is the default behavior.
-w	Indicates that lines of text that are too long to be displayed on one line in the clipboard should wrap around to the following lines.

Resources

xclipboard understands all of the Core resource names and classes, as well as the following application-specific resource:

`wrap` (class `Wrap`)
: If `True`, lines of text that are too long to be displayed on one line in the clipboard will wrap around to the following lines. If `False` (the default), long lines will not wrap.

Sending and Retrieving Clipboard Contents

Text is copied *to* the clipboard whenever a client asserts ownership of the CLIPBOARD selection. Text is copied *from* the clipboard whenever a client requests the contents of the CLIPBOARD selection. This doesn't necessarily happen automatically; you must add translations for each application that you want to have work with *xclipboard*. Examples of event bindings that a user may wish to include in a resource configuration file to use the clipboard from *xterm* are:

```
*VT100.Translations: #override \n\
     Button1 <Btn3Down>: select-end(PRIMARY,CUT_BUFFER0,CLIPBOARD)\n\
     !Shift <Btn2Up>:       insert-selection(CLIPBOARD)\n\
     ~Shift ~Ctrl ~Meta <Btn2Up>:   insert-selection(PRIMARY,CUT_BUFFER0)
```

The first translation, `Button1 <Btn3Down>: select-end(CLIPBOARD,CUT_BUFFER0,CLIPBOARD)`, specifies that if button 3 is pressed while button 1 is held down, the selection will be made the PRIMARY selection, copied to CUT_BUFFER0, and added to the CLIPBOARD. If button 3 isn't pressed while button 1 is held down, the default *xterm* translation, namely to add the selection to the PRIMARY selection and CUT_BUFFER0 on any key up, takes effect instead.

The second translation line specifies a way to paste the CLIPBOARD selection (the current contents of the *xclipboard* window): by holding the Shift key and clicking the second pointer button.

The third translation pastes the contents of the PRIMARY selection, or if that is empty, CUT_BUFFER0. `~Ctrl` is specified to keep this translation from conflicting with the translations that invoke the *xterm* menus; `~Meta` prevents a conflict with *twm* functions. We've added `~Shift` to prevent a conflict with the action that pastes the CLIPBOARD selection.

Widget Hierarchy

In order to specify resources, it is useful to know the hierarchy of the widgets that compose *xclipboard*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name. The first line shows the application class and instance names:

```
XClipboard  xclipboard
        Form  form
                Command  quit
                Command  delete
                Command  new
```

```
Command  next
Command  prev
Text     text
```

For information on the resources available in each of the Athena widgets, see Appendix G, *Widget Resources*.

Files

/usr/lib/X11/app-defaults/XClipboard Specifies required resources.

See Also

X, xterm; Chapter 5, *The xterm Terminal Emulator*; Chapter 11, *Setting Resources*; individual client reference pages for the appropriate translations to send selections to the CLIPBOARD.

Authors

Ralph R. Swick, DEC/MIT Project Athena;
Chris Peterson, MIT X Consortium;
Keith Packard, MIT X Consortium.

xclock

Analog/Digital Clock

Name

xclock – continuously display the time in either analog or digital form.

Syntax

xclock [*options*]

Description

xclock continuously displays the time of day, either in digital or analog form. In digital form, *xclock* displays the time using a 24-hour clock. It also displays the day, month, and year. In analog form, *xclock* displays a standard 12-hour clock face. You can set up more than one clock simultaneously.

The default clock is an analog clock with a black foreground on a white background. If you want to change the clock's appearance, type in the appropriate options. For example,

```
% xclock -bd slateblue -fg navyblue -hl darkslategray &
```

sets up a conventional 12-hour clock with a slate blue window border, navy blue tick marks, and dark slate gray hands.

By default, the clock is positioned in the upper-left corner of your background window. If you are running the default version of *mwm*, the window manager will place the clock in the upper-left quadrant of the screen, offset from the corner.

Options

xclock accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. (We've included some of the more commonly used Toolkit options later in this section.) In addition, *xclock* accepts the following application-specific options:

`-help` Displays a brief summary of *xclock*'s calling syntax and options.

`-analog`
: Draws a conventional 12-hour clock face with tick marks for each minute and stroke marks for each hour. This is the default.

`-digital` or `-d`
: Displays the date and time in digital format. Note that `-display` must be used to specify a display.

`-chime`
: Indicates that the clock should chime once on the half hour and twice on the hour.

`-hd` *color*
: Specifies the color of the hands on an analog clock. The default is black.

`-hl` *color*
: Specifies the color of the edges of the hands on an analog clock. Only useful on color displays. The default is black.

`-padding` *`pixels`*
: Specifies the width in pixels of the space between the window border and any portion of the *xclock* display. The default is 10 pixels in digital mode and 8 pixels in analog mode.

`-update` *`seconds`*
: Specifies the frequency in seconds with which *xclock* updates its display. If the *xclock* window is obscured and then exposed, *xclock* overrides this setting and redisplays immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

The following standard X Toolkit options are commonly used with *xclock*:

`-bw` *`pixels`*
: Specifies the width in pixels of the border around the *xclock* window. The default is 2 pixels.

`-fg` *`color`*
: Determines the color of the text in digital mode, and the color of the tick and stroke marks in analog mode. The default is black.

`-fn` *`font`*
: Specifies the font to be used in digital mode. Any fixed-width font may be used. The default is `6x10`.

`-geometry` *`geometry`*
: Sets *xclock* window size and location according to the geometry specification. The `-geometry` option can be (and often is) abbreviated to `-g`, unless there is a conflicting option that begins with "g". The argument to the geometry option (*`geometry`*) is referred to as a "standard geometry string," and has the form *`width`*`x`*`height`*`±`*`xoff`*`±`*`yoff`*.

  In digital mode, height and width are determined by the font in use, unless otherwise specified. In analog mode, width and height defaults are 164 pixels, unless otherwise specified. The default value for any unspecified x or y offset is `-0`. All values are in pixels. If you do not specify the geometry, the window manager may place the window; if not, *xclock* will ask you for placement when it starts up.

`-display` [*`host`*]`:`*`server`*[`.`*`screen`*]
: Allows you to specify the physical display, server, and screen on which to create the *xclock* window. See "Options" on the *X* reference page for an example of usage.

  Note that `-display` cannot be abbreviated to `-d`, which is shorthand for *xclock*'s `-digital` option.

`-xrm` *`resourcestring`*
: Specifies a resource string to be used. This is especially useful for setting resources that do not have separate command-line options.

Resources

xclock uses the Athena Clock widget. It understands all of the Core resource names and classes as well as the new resources defined by the Clock widget. Resources you may want to set in user resource files include:

`analog` (class `Boolean`)
: Specifies whether or not an analog clock should be used instead of a digital one. The default is `true`.

`chime` (class `Boolean`)
: Specifies whether or not a bell should be rung on the half hour and on the hour. The default is `false`.

`hands` (class `Foreground`)
: Specifies the color of the insides of the clock's hands. The default is the `foreground` color.

`highlight` (class `Foreground`)
: Specifies the color used to highlight the clock's hands. The default is the `foreground` color.

`padding` (class `Margin`)
: Specifies the amount of internal padding in pixels to be used. The default is 8.

`update` (class `Interval`)
: Specifies the frequency in seconds at which the time should be redisplayed.

You may also want to set the following X Toolkit resources:

`background` (class `Background`)
: Determines the background color. The default is white.

`font` (class `Font`)
: Specifies the font to be used for the digital clock. Note that variable-width fonts currently will not always display correctly.

`foreground` (class `Foreground`)
: Specifies the color for the tick marks and stroke marks. Using the class specifies the color for all things that normally would appear in the foreground color. The default is black since the core default for background is white.

`height` (class `Height`)
: Specifies the height of the clock.

`reverseVideo` (class `ReverseVideo`)
: Specifies that the foreground and background colors should be reversed.

`width` (class `Width`)
: Specifies the width of the clock.

Widget Hierarchy

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xclock*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XClock  xclock
     Clock  clock
```

Files

/usr/lib/X11/app-defaults/XClock	Specifies default resources.

Bugs

xclock believes the system clock.

When in digital mode, the string should be centered automatically.

There should be a way to exit the program.

See Also

X, oclock, xrdb, time(3C); Chapter 8, *Other Clients*; Appendix G, *Widget Resources*.

Authors

Tony Della Fera (MIT-Athena, DEC);
Dave Mankins (MIT-Athena, BBN);
Ed Moy (UC Berkeley).

xcmsdb

— Edit Screen Color Properties —

Name

xcmsdb – Xlib screen color characterization data utility.

Syntax

xcmsdb [*options*] [*filename*]

Description

xcmsdb is used to load, query, or remove Screen Color Characterization Data stored in properties on the root window of the screen. Screen Color Characterization Data is an integral part of Xlib, necessary for proper conversion between device-independent and device-dependent color specifications. Xlib uses the XDCCC_LINEAR_RGB_MATRICES and XDCCC_LINEAR_RGB_CORRECTION properties to store color characterization data for color monitors. It uses XDCCC_GRAY_SCREENWWHITEPOINT and XDCCC_GRAY_CORRECTION properties for gray scale monitors. Because Xlib allows the addition of Screen Color Characterization Function Sets, added function sets may place their Screen Color Characterization Data on other properties. This utility is unaware of these other properties; therefore, you will need to use a similar utility provided with the function set, or use the *xprop* utility.

The ASCII readable contents of *filename* (or the standard input if no input file is given) are appropriately transformed for storage in properties, provided the `-query` or `-remove` options are not specified.

Options

xcmsdb accepts the following options:

`-query` Specifies that the XDCCC properties be read off of the screen's root window. If successful, this transforms the data into a more readable format, then sends the data to standard output.

`-remove` Attempts to remove the XDCCC properties on the screen's root window.

`-color` Sets the query and remove options to only check for the XDCCC_LINEAR_RGB_MATRICES and XDCCC_LINEAR_RGB_CORRECTION properties. If the `-color` option is not set then the query and remove options check for all the properties.

`-format 32 | 16 | 8`
Specifies the property format (32, 16, or 8 bits per entry) for the XDCCC_LINEAR_RGB_CORRECTION property. Precision of encoded floating point values increases with the increase in bits per entry. The default is 32 bits per entry.

See Also

xprop; Chapter 12, *Specifying Color*; Volume One, *Xlib Programming Manual*; Volume Two, *Xlib Reference Manual.*

Copyright

Author

Chuck Adams, Tektronix Inc.

xcol

Name

xcol – display colors and change color entries in resource files.

Syntax

xcol [*options*] [*filename*]

Description

The public domain *xcol* program displays the colors defined in the *rgb.txt* file of the X server. The colors are sorted by their names and their RGB-values and shown in a cube in the ColorView window (The positions represent the RGB-values). Since there usually are more colors defined in the file than cells in the colormap, entries with the same name, but different RGB-values for different intensities, are grouped together.

If a filename is given as a parameter, all occurrences of color names in that file are shown in a second window, called the TextView window. To change the colors specified in the file, a text line has to be made active. Then a new color can be selected in the ColorView window.

To get a better impression of the color, a help (highlight/background) color can be selected for each text line. If two lines define a foreground and a background color, an association can be made and the colors of both lines can be selected together.

Note that as of the printing of this guide, *xcol* had not been updated for Release 5, but our testing showed it could run under the standard X11R5 server. See Chapter 8, *Other Clients*, for a tutorial.

Pointer Commands

In the ColorView window, pointer clicks have the following effects:

First button: Selects color for the active resource line in the TextView window.

Second button: Selects help (i.e., highlight) color for the active resource line in the TextView window.

Third button: Moves pointer to the pixel of the color specified by the active resource line in the TextView window.

In the TextView window, pointer clicks have the following effects:

First button: Selects the text line as the active line.

Second button: Toggles reverse video mode or connects/disconnects two associated lines (e.g., background and foreground color specifications for the same resource).

Third button: Highlights the line in the text file.

Options

`-rv`

Color positions are reversed in the cube. Some new colors can become visible in the area of the very bright colors.

`-b`*`number`*
The size of color blocks is set to the constant *number*. By default, it depends on the size of the ColorView window.

`-gran`*`number`*
Sets the maximum number of associated colors to *number*. By default, this value is 11. (You can see the effect when reaching the grey field, where 101 associated entries are possibly available.)

`-dark`*`number`*
Sets the percentage of the intensity of other colors. The default is 50. It's easier to select a color if the screen is darkened a little bit.

`+`*`char_list`*
Adds characters in *char_list* to list of "space" characters. A color string in the text file must occur between 2 'space' characters to be recognized. By default, these characters are " ", "\t", "\n", ":", "'".

`-strings`
Specifies that names of colors in the file are only recognized when used in a string (useful for C source code). This means that the list of "space" characters is set to "'" only.

`-case`
Specifies that all occurrences of color names in the wrong case are replaced by the appropriate color names from the *rgb.txt* file.

`-help`
Prints some instructions (e.g., on the usage of buttons).

`-file` *`filename`*
Specifies an alternative file to the *rgb.txt* file.

Files

/usr/lib/X11/rgb.txt

See Also

X; Chapter 8, *Other Clients*; Chapter 12, *Specifying Color*.

Bugs

The author wanted the program to work with the selection mechanism used in *xterm* (when no text file is given), but it seems to be too complicated. So, the current version always stores the string of the color in CUT_BUFFER0. If anyone has an easy way to use the correct selection mechanism, please contact the author.

Copyright

Author

Helmut Hoenig (hoenig@informatik.uni-kl.de).

xcoloredit

Name

xcoloredit – find RGB color values by graphical color mixing.

Syntax

xcoloredit [*options*] [{0-255} ..]

Description

xcoloredit is a public domain client that provides a graphical method of mixing the three primary colors available on a color workstation. This mixing can be done using the Red, Green, and Blue slider controls on the left of the window or using the Hue, Saturation, and Value slider controls on the right.

The three boxes above the Red, Green, and Blue slider controls are used for linking the controls together via the fourth slider to the right of the blue slider. While in the slider controls the first mouse button increments the color components value, the third mouse button decrements the value (this only works with the Red, Green, Blue, and Linked sliders). The middle mouse button allows the user to continually change the value.

The results of the color mixing is shown in the four central squares. The three smaller squares shows the intensities of the red, green, and blue components. The hexidecimal value below these squares is the corresponding color value which can be used in defaults files. This value is also placed in the PRIMARY_COLOR selection property. If the user presses the color value button, the button is highlighted and the color value is placed in PRIMARY selection as well (useful for pasting into defaults files).

At the bottom of the main window are 36 color cells. The current color cell is highlighted by a box drawn around it. By clicking with the first mouse button in another cell this new cell's current value can be edited (if the cell has no defined value, the current cells value is copied to it and the cell is highlighted with a dashed box). These color cells can be connected to cells in the default colormap of the display. To do this the user must give the colormap entry number(s) (pixel number) as command-line argument(s).

The text shown in the mixed color window can be displayed in one of the 36 color cell colors. Typing "c" or "t" in this window changes the color of the text to that of the currently selected color cell. This can be used to see what text will look like with different foreground and background colors. The example text can be modified using the *-text* command-line option.

Options

xcoloredit accepts the standard X Toolkit options, which are listed on the *X* reference page. In addition, *xcoloredit* accepts the following application-specific options:

`-format`

Specifies the format used to display the RGB value of the color. This format is used by the *printf*(2) function call. By default the format is set to "#%02x%02x%02x" which is the standard RGB format for X.

`-silent`
Specifies that the edited color values are not printed out when *xcoloredit* quits.

`-text`
Sets the example text to display in the mixed color window. Newlines are allowed in this string.

Selection Atoms

The following selection atoms are used/defined:

PRIMARY_COLOR
Current color selection value.

PRIMARY
Current color selection value when highlighted.

See Also

xtici; Chapter 8, *Other Clients*; Chapter 12, *Specifying Color*.

Author

Richard Hesketh, University of Kent at Canterbury, March 1989.
rlh2@ukc.ac.uk

xconsole

— Console Message Monitor

Name

xconsole – monitor system console messages.

Syntax

xconsole [*options*]

Description

xconsole displays system messages that are usually sent to */dev/console*. See Appendix A, *Managing Your Environment*, for an example of usage.

Options

xconsole recognizes all of the standard X Toolkit options, which are listed on the *X* reference page. (See Chapter 10, *Command-line Options*, for more information.) *xconsole* also recognizes the following application-specific options:

`-daemon`
: Causes *xconsole* to place itself in the background, using fork/exit.

`-exitOnFail`
: When set, this option directs *xconsole* to exit when it is unable to redirect the console output.

`-file` *filename*
: To monitor some other device, use this option to specify the device name. This does not work on regular files as they are always ready to be read from.

`-notify` or `-nonotify`
: When new data is received from the console and the notify option is set, the icon name of the application has "*" appended, so that the occurrence of a message is evident even though the application is iconified. `-notify` is the default, although this feature does not seem to work reliably in all environments.

  Note that if you're running *mwm* and using an Icon Box, the asterisk cannot be appended to the icon name.

`-stripNonprint`
: Causes nonprinting characters from the console output to be stripped before messages appear in the *xconsole* window. This is the default.

`-verbose`
: When set, this option directs *xconsole* to display an informative message in the first line of the text buffer.

Resources

xconsole recognizes the Core resource names and classes. (See Appendix G, *Widget Resources*, for more information.) In addition, *xconsole* defines the following application resources:

`daemon` (class `Daemon`)
: If `True`, causes *xconsole* to place itself in the background, using fork/exit. `False` by default.

`exitOnFail` (class `ExitOnFail`)
: If `True`, directs *xconsole* to exit when it is unable to redirect the console output. `False` by default.

`file` (class `File`)
: To monitor some other device, use this resource to specify the device name. This does not work on regular files as they are always ready to be read from.

`notify` (class `Notify`)
: When new data is received from the console and the `notify` resource is `True`, the icon name of the application has "*" appended, so that it is evident even when the application is iconified. `False` suppresses this behavior. `True` by default. (Note, however, that this feature does not seem to work reliably in all environments.)

`stripNonprint` (class `StripNonprint`)
: When `True`, causes nonprinting characters from the console output to be stripped before messages appear in the *xconsole* window. `True` by default.

`verbose` (class `Verbose`)
: When `True`, directs *xconsole* to display an informative message in the first line of the text buffer. `False` by default.

Widget Hierarchy

In order to specify resources, it is useful to know the hierarchy of the widgets that compose *xconsole*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name:

```
XConsole  xconsole
        XConsole  text
```

Notice that *xconsole* uses the Athena Text widget. See Appendix G, *Widget Resources*, and the Athena Widget Set documentation in the standard distribution.

Files

/usr/lib/X11/app-defaults/XConsole
: Specifies required resources.

See Also

X, xrdb; Appendix A, *Managing Your Environment*; Appendix G, *Widget Resources*.

Bugs

When *xconsole* is iconified (and program defaults apply), an asterisk should be appended to the icon name (see `-notify` and the `notify` resource), but this does not happen consistently in every environment.

Author

Keith Packard (MIT X Consortium).

xcrtca

Name

xcrtca – CRT color analyzer driver.

Syntax

xcrtca [*options*]

Description

xcrtca is used in conjunction with *xsccd* to create a color database for the Xcms Color Management System using either the Minolta CA-100 (default) or Tektronix J17, low cost colorimeters.

The Minolta CA-100 in use at the MIT X Consortium includes the Low Luminance option which is necessary to get the required accuracy for readings at low luminance levels. The CA-100 outputs measurements in the CIExyY color space at a baud rate of 9600.

The output format for the Tektronix J17 is selectable on the device as well as the baud rate. Select either CIExyY or CIEXYZ format for use with this program. The default baud rate for the J17 is 2400. Note that the default baud rate for this program is 9600; therefore, unless you select 9600 on the J17, you will need to specify the baud rate with the `-baud` option.

Options

`-baud` *baudrate*
: Baud rate of the colorimeter. Default for this program is 9600.

`-delay` *seconds*
: The amount of delay between the change of the color patch and the first reading of each primary color. Default is 4 seconds.

`-device` *device_name*
: Specifies the name of the colorimeter. Acceptable values: either `ca100` (the default) or `j17`.

`-file` *filename*
: Specifies the filename for the results; otherwise results are sent to standard output.

`-format` [`xyY`| `XYZ`]
: Format of measurements received from the colorimeter.

`-input` *pathname*
: Specifies the pathname of the *tty*. Default for this program is */dev/ttya*.

`-model` *model_string*
: Specifies the model of the CRT.

`-name` *name_string*
: Specifies the name (e.g., manufacturer) to be associated with the CRT.

Caveats

This program has been coded for a Sun SparcStation and for a default visual with 8 bits_per_rgb.

See Also

xcmsdb, xsccd; Chapter 12, *Specifying Color*.

Authors

Dave Sternlicht, Keith Packard, MIT X Consortium;
Al Tabayoyon, Chuck Adams, Tektronix Inc.

xditview

Name

xditview – display *ditroff* DVI files.

Syntax

xditview [*options*] [*filename*]

Description

The *xditview* program displays *ditroff* output on an X display. As of Release 5, it uses no special font metrics and automatically converts the printer coordinates into screen coordinates, using the user-specified screen resolution, rather than the actual resolution so that the appropriate fonts can be found. If "–" is given as the *filename*, *xditview* reads from standard input. If "|" is the first character of the *filename*, *xditview* forks *sh* to run the rest of the *filename* and uses the standard output of that command.

Options

xditview accepts all of the standard X Toolkit command-line options, which are listed on the *X* reference page. (We've included one of the more commonly used Toolkit options later in this section.) In addition, *xditview* accepts the following application-specific options:

-backingStore *backing_store_type*
: Redisplay of the DVI window can take up to a second or so. This option causes the server to save the window contents so that when it is scrolled around the viewport, the window is painted from contents saved in backing store. *backing_store_type* can be one of Always, WhenMapped, or NotUseful.

-noPolyText
: Some X servers incorrectly implement PolyText with multiple strings per request. This option suppresses the use of this feature in *xditview*. (Available as of Release 5.)

-page *page_number*
: Specifies the page number of the document to be displayed when the client is started.

-resolution *dots_per_inch*
: Specifies the desired screen resolution to use. Fonts will be opened by requesting this resolution field in the XLFD names. (Available as of Release 5.)

The following standard X Toolkit option is commonly used with *xditview*:

-fn *font*
: Specifies the font to be used for displaying widget text. The default is *fixed*.

Resources

This program uses a Dvi widget. It understands all of the core resource names and classes as well as the following:

font (class Font)
: Specifies the font to be used for error messages.

`fontMap` (class `FontMap`)

To associate the *ditroff* fonts with appropriate X fonts, this string resource contains a set of newline-separated specifications, each of which consists of a *ditroff* name, some white space and an XLFD pattern with wildcard (*) characters in appropriate places to allow all sizes to be listed. This resource has been added in Release 5. The default `fontMap` is:

```
R  -*-times-medium-r-normal--*-*-*-*-*-*-iso8859-1\n\
I  -*-times-medium-i-normal--*-*-*-*-*-*-iso8859-1\n\
B  -*-times-bold-r-normal--*-*-*-*-*-*-iso8859-1\n\
F  -*-times-bold-i-normal--*-*-*-*-*-*-iso8859-1\n\
BI -*-times-bold-i-normal--*-*-*-*-*-*-iso8859-1\n\
C  -*-courier-medium-r-normal--*-*-*-*-*-*-iso8859-1\n\
CO -*-courier-medium-o-normal--*-*-*-*-*-*-iso8859-1\n\
CB -*-courier-bold-r-normal--*-*-*-*-*-*-iso8859-1\n\
CF -*-courier-bold-o-normal--*-*-*-*-*-*-iso8859-1\n\
H  -*-helvetica-medium-r-normal--*-*-*-*-*-*-iso8859-1\n\
HO -*-helvetica-medium-o-normal--*-*-*-*-*-*-iso8859-1\n\
HB -*-helvetica-bold-r-normal--*-*-*-*-*-*-iso8859-1\n\
HF -*-helvetica-bold-o-normal--*-*-*-*-*-*-iso8859-1\n\
N  -*-new century schoolbook-medium-r-normal--*-*-*-*-*-*-iso8859-1\n\
NI -*-new century schoolbook-medium-i-normal--*-*-*-*-*-*-iso8859-1\n\
NB -*-new century schoolbook-bold-r-normal--*-*-*-*-*-*-iso8859-1\n\
NF -*-new century schoolbook-bold-i-normal--*-*-*-*-*-*-iso8859-1\n\
A  -*-charter-medium-r-normal--*-*-*-*-*-*-iso8859-1\n\
AI -*-charter-medium-i-normal--*-*-*-*-*-*-iso8859-1\n\
AB -*-charter-bold-r-normal--*-*-*-*-*-*-iso8859-1\n\
AF -*-charter-bold-i-normal--*-*-*-*-*-*-iso8859-1\n\
S  -*-symbol-medium-r-normal--*-*-*-*-*-*-adobe-fontspecific\n\
S2 -*-symbol-medium-r-normal--*-*-*-*-*-*-adobe-fontspecific\n
```

`foreground` (class `Foreground`)

Specifies the default foreground color.

`pageNumber` (class `PageNumber`)

Specifies the page number to be displayed at startup.

Using xditview with ditroff

You can use any DVI file with *xditview*, although DVI files that use the fonts appropriate to the `fontMap` will look more accurate on the screen. On servers that support scaled fonts, all requested font sizes will be accurately reflected on the screen; for servers that do not support scaled fonts, *xditview* will use the closest font from the same family.

Files

/usr/lib/X11/app-defaults/Xditview
Specifies required resources.

/usr/lib/X11/app-defaults/Xditview-chrtr
Specifies the default `fontMap`.

See Also

X, xrdb, ditroff(1); the paper entitled "X Logical Font Description Conventions."

Authors

Portions of this program originated in *xtroff* which was derived from *suntroff*.

Keith Packard (MIT X Consortium);
Richard L. Hyde (Purdue);
David Slattengren (Berkeley);
Malcom Slaney (Schlumberger Palo Alto Research);
Mark Moraes (University of Toronto).

xdm

Name

xdm – X display manager.

Syntax

xdm [*options*]

Description

xdm manages a collection of X displays that may be on the local host or remote servers. The design of *xdm* was guided by the needs of X terminals as well the X Consortium standard XDMCP, the X Display Manager Control Protocol. *xdm* provides services similar to those provided by *init*, *getty*, and *login* on character terminals: prompting for login name and password, authenticating the user, and running a "session."

A *session* is defined by the lifetime of a particular process; in the traditional character-based terminal world, it is the user's login shell process. In the *xdm* context, it is an arbitrary session manager. This is because, in a windowing environment, a user's login shell process does not necessarily have any terminal-like interface with which to connect. When a real session manager is not available, a window manager or terminal emulator is typically used as the "session manager," meaning that termination of this process terminates the user's session.

When the session is terminated, *xdm* resets the X server and (optionally) restarts the whole process.

When *xdm* receives an Indirect query via XDMCP, it can run a *chooser* process to perform an XDMCP BroadcastQuery (or an XDMCP Query to specified hosts) on behalf of the display, and offer a menu of possible hosts that offer XDMCP display management. This feature is useful with X terminals that do not offer a host menu themselves.

Because *xdm* provides the first interface that users will see, it is designed to be simple to use and easy to customize to the needs of a particular site. *xdm* has many options, most of which have reasonable defaults. Browse through the various sections, picking and choosing the things you want to change. Pay particular attention to "The Xsession Program," which will describe how to set up the style of session desired.

Options

Note that all of these options, except `-config`, specify values that can also be specified in the configuration file as resources.

`-config` *configuration_file*

Specifies the configuration file which specifies resources to control the behavior of *xdm* parameters. The default is */usr/lib/X11/xdm/xdm-config*.

`-daemon`

Specifies `true` as the value for the `DisplayManager.daemonMode` resource. This makes *xdm* close all file descriptors, disassociate the controlling terminal, and put itself in the background when it first starts up (just like the host of other daemons). It is the default behavior.

`-debug` *`debug_level`*
: Specifies the numeric value for the `DisplayManager.debugLevel` resource. A non-zero value causes *xdm* to print lots of debugging statements to the terminal; it also disables the `DisplayManager.daemonMode` resource, forcing *xdm* to run synchronously. To interpret these debugging messages, a copy of the source code for *xdm* is almost a necessity. No attempt has been made to rationalize or standardize the output.

`-error` *`error_log_file`*
: Specifies the value for the `DisplayManager.errorLogFile` resource. This file contains errors from *xdm* as well as anything written to standard error by the various scripts and programs run during the progress of the session.

`-nodaemon`
: Specifies `false` as the value for the `DisplayManager.daemonMode` resource. This suppresses the normal daemon behavior, which is for *xdm* to close all file descriptors, disassociate itself from the controlling terminal, and put itself in the background when it first starts up.

`-resources` *`resource_file`*
: Specifies the value for the `DisplayManager*resources` resource. This file is loaded using *xrdb* to specify configuration parameters for the authentication widget.

`-server` *`server_entry`*
: Specifies the value for the `DisplayManager.servers` resource. (See the section "Server Specification.")

`-udpPort` *`port_number`*
: Specifies the value for the `DisplayManager.requestPort` resource. This sets the port number which *xdm* will monitor for XDMCP requests. As XDMCP uses the registered well-known UDP port 177, this resource should probably not be changed except for debugging.

`-session` *`session_program`*
: Specifies the value for the `DisplayManager*session` resource. This indicates the program to run as the session when the user logs in.

`-xrm` *`resource_specification`*
: Allows an arbitrary resource to be specified, just as most toolkit applications.

Resources

At many stages the actions of *xdm* can be controlled through the use of the configuration file, which is in the X resource format. See Chapter 11 for a description of the resource file format. Some resources modify the behavior of *xdm* on all displays, while others modify its behavior on a single display. Where actions relate to a specific display, the display name is inserted into the resource name between "DisplayManager" and the final resource name segment. For example, `DisplayManager.expo_0.startup` is the name of the resource that defines the startup shell file on the "expo:0" display. Because the resource manager uses colons to

separate the name of the resource from its value and dots to separate resource name parts, *xdm* substitutes underscores for both dots and colons when generating the resource name.

`DisplayManager.authDir`
: Names a directory in which *xdm* stores authorization files while initializing the session. The default value is */usr/lib/X11/xdm*.

`DisplayManager.errorLogFile`
: Error output is normally directed at the system console. To redirect it, set this resource to any filename. A method to send these messages to *syslog* should be developed for systems that support it; however the wide variety of interfaces precludes any system-independent implementation. This file also contains any output directed to standard error by *Xsetup*, *Xstartup*, *Xsession*, and *Xreset*, so it will contain descriptions of problems in those scripts as well.

`DisplayManager.debugLevel`
: If the integer value of this resource is greater than zero, reams of debugging information will be printed. It also disables daemon mode, which would redirect the information into the bit-bucket and allows non-root users to run *xdm*, which would normally not be useful.

`DisplayManager.daemonMode`
: Normally, *xdm* attempts to make itself into a daemon process unassociated with any terminal. This is accomplished by forking and leaving the parent process to exit, then closing file descriptors and releasing the controlling terminal. When attempting to debug *xdm*, this is quite bothersome. Setting this resource to `false` will disable this feature.

`DisplayManager.pidFile`
: The filename specified will be created to contain an ASCII representation of the process ID of the main *xdm* process. *xdm* also uses file locking on this file to attempt to eliminate multiple daemons running on the same machine, which would cause quite a bit of havoc.

`DisplayManager.lockPidFile`
: Controls whether *xdm* uses file locking to keep multiple display managers (*xdm* processes) from running amok. On System V, this uses the *lockf* library call, while on BSD it uses *flock*.

`DisplayManager.autoRescan`
: This Boolean controls whether *xdm* rescans the configuration, access control, and authentication keys files after a session terminates and the files have changed. By default it is `True`. You can force *xdm* to reread these files by sending a SIGHUP to the main process.

`DisplayManager.removeDomainname`
: When computing the display name for XDMCP clients, the name resolver will typically create a fully qualified hostname of the terminal. Since this is sometimes

confusing, *xdm* will remove the domain name portion of the hostname if it is the same as the domain name for the local host when this variable is set. By default the value is `True`.

`DisplayManager.keyFile`
: XDM-AUTHENTICATION-1 style XDMCP authentication requires that a private key be shared between *xdm* and the terminal. This resource specifies the file containing those values. Each entry in the file consists of a display name and the shared key. By default, *xdm* does not include support for XDM-AUTHENTICATION-1 as it requires DES, which is not generally distributable because of United States export restrictions.

`DisplayManager.accessFile`
: To prevent unauthorized XDMCP service and to allow forwarding of XDMCP Indirect-Query requests, this file contains a database of hostnames which are either allowed direct access to this machine, or have a list of hosts to which queries should be forwarded. The format of this file is described in the section "XDMCP Access Control." (Available as of Release 5.)

`DisplayManager.exportList`
: A whitespace-separated list of additional environment variables to pass on to the *Xsetup*, *Xstartup*, *Xsession*, and *Xreset* programs. (Available as of Release 5.)

`DisplayManager.randomFile`
: A file to checksum to generate the seed of authorization keys. This should be a file that changes frequently. The default is */dev/mem*. (Available as of Release 5.)

`DisplayManager.`*`DISPLAY`*`.resources`
: Specifies the name of the file to be loaded by *xrdb* as the resource database onto the root window of screen 0 of the display. The *Xsetup* program, the Login widget, and *chooser* will use the resources set in this file. This resource database is loaded just before the authentication procedure is started, so it can control the appearance of the "login" window. See "Authentication Widget Resources," which describes the various resources which are appropriate to place in this file. There is no default value for this resource, but the conventional name is */usr/lib/X11/xdm/Xresources*.

`DisplayManager.`*`DISPLAY`*`.chooser`
: Specifies the program run to offer a host menu for Indirect queries redirected to the special hostname CHOOSER. */usr/lib/X11/xdm/chooser* is the default. See the sections "XDMCP Access Control" and "Chooser." (Available as of Release 5.)

`DisplayManager.`*`DISPLAY`*`.setup`
: This specifies a program which is run (as root) before offering the Login window. This may be used to change the appearance of the screen around the Login window or to put up other windows (e.g., you may want to run *xconsole* here). By default, no program is run. The conventional name for a file used here is *Xsetup*. See the section "The Setup Program." (Available as of Release 5.)

`DisplayManager.`*`DISPLAY`*`.xrdb`
: Specifies the program used to load the resources. By default, *xdm* uses */usr/bin/X11/xrdb*.

`DisplayManager.`*`DISPLAY`*`.cpp`
: Specifies the name of the C preprocessor used by *xrdb*.

`DisplayManager.`*`DISPLAY`*`.openDelay,`
`DisplayManager.`*`DISPLAY`*`.openRepeat,`
`DisplayManager.`*`DISPLAY`*`.openTimeout,`
`DisplayManager.requestPort`
: Indicates the UDP port number that *xdm* uses to listen for incoming XDMCP requests. Unless you need to debug the system, leave this with its default value of 177.

`DisplayManager.`*`DISPLAY`*`.reset`
: Specifies a program which is run (as root) after the session terminates. Again, by default, no program is run. The conventional name is *Xreset*. See "The Xreset Program" below.

`DisplayManager.servers`
: Specifies either a filename full of server entries, one per line (if the value starts with a slash), or a single server entry. See the section "Server Specification" for a description of this resource.

`DisplayManager.`*`DISPLAY`*`.session`
: Specifies the session to be executed (not running as root). By default, */usr/bin/X11/xterm* is run. The conventional name is *Xsession*. See "The Xsession Program."

`DisplayManager.`*`DISPLAY`*`.startAttempts`
: Numeric resources control the behavior of *xdm* when attempting to open intransigent servers. `openDelay` is the length of the pause (in seconds) between successive attempts. `openRepeat` is the number of attempts to make. `openTimeout` is the amount of time to wait while actually attempting the open (i.e., the maximum time spent in the *connect* system call). `startAttempts` is the number of times this entire process is done before giving up on the server. After `openRepeat` attempts have been made, or if `openTimeout` seconds elapse in any particular attempt, *xdm* terminates and restarts the server, attempting to connect again. This process is repeated `startAttempts` times, at which point the display is declared dead and disabled. Although this behavior may seem arbitrary, it has been empirically developed and works quite well on most systems. The default values are 5 for `openDelay`, 5 for `openRepeat`, 30 for `openTimeout`, and 4 for `startAttempts`.

`DisplayManager.`*`DISPLAY`*`.startup`
: Specifies a program which is run (as root) after the authentication process succeeds. By default, no program is run. The conventional name for a file used here is *Xstartup*. See "The Xstartup Program" below.

`DisplayManager.`*`DISPLAY`*`.pingInterval`
`DisplayManager.`*`DISPLAY`*`.pingTimeout`

To discover when remote displays disappear, *xdm* occasionally "pings" them, using an X connection and sending `XSync` calls. `pingInterval` specifies the time (in minutes) between each ping attempt; `pingTimeout` specifies the maximum amount of time (in minutes) to wait for the terminal to respond to the request. If the terminal does not respond, the session is declared dead and terminated. By default, both are set to 5 minutes. If you frequently use X terminals that can become isolated from the managing host, you may wish to increase this value. The only worry is that sessions will continue to exist after the terminal has been accidentally disabled. *xdm* will not ping local displays. Although it would seem harmless, it is unpleasant when the workstation session is terminated as a result of the server hanging for NFS service and not responding to the ping.

`DisplayManager.`*`DISPLAY`*`.terminateServer`

Specifies whether the X server should be terminated when a session terminates (instead of resetting it). This option can be used when the server tends to grow without bound over time in order to limit the amount of time the server is run. The default value is `false`.

`DisplayManager.`*`DISPLAY`*`.userPath`

xdm sets the PATH environment variable for the session to this value. It should be a colon separated list of directories; see *sh*(1) for a full description. The default value can be specified at build time in the X system configuration file with `DefUserPath`; *:/bin:/usr/bin:/usr/bin/X11:/usr/ucb* is a common value.

`DisplayManager.`*`DISPLAY`*`.systemPath`

xdm sets the PATH environment variable for the startup and reset scripts to the value of this resource. The default for this resource is specified at build time with the `DefaultSystemPath` entry in the system configuration file; a common choice is */etc:/bin:/usr/bin:/usr/bin/X11:/usr/ucb* . Note the absence of "." from this entry. This is a good practice to follow for root; it avoids many common Trojan horse system penetration schemes.

`DisplayManager.`*`DISPLAY`*`.systemShell`

xdm sets the SHELL environment variable for the startup and reset scripts to the value of this resource. By default, it is */bin/sh*.

`DisplayManager.`*`DISPLAY`*`.failsafeClient`

If the default session fails to execute, *xdm* will fall back to this program. This program is executed with no arguments, but executes using the same environment variables as the session would have had. See "The Xsession Program." By default, */usr/bin/X11/xterm* is used.

`DisplayManager.`*`DISPLAY`*`.grabServer`
`DisplayManager.`*`DISPLAY`*`.grabTimeout`

To improve security, *xdm* grabs the server and keyboard while reading the login name and password. The `grabServer` resource specifies if the server should be held for the duration of the login name and password reading: when `false`, the server is ungrabbed after the keyboard grab succeeds; otherwise, the server is grabbed until just before the session begins. The default is `false`. The `grabTimeout` resource specifies the maximum time *xdm* will wait for the grab to succeed. The grab may fail if some other client has the server grabbed, or possibly if the network latencies are very high. This resource has a default value of 3 seconds; you should be cautious when raising it, as a user can be spoofed by a look-alike window on the display. If the grab fails, *xdm* kills and restarts the server (if possible) and the session.

`DisplayManager.`*`DISPLAY`*`.authorize`
`DisplayManager.`*`DISPLAY`*`.authName`

`authorize` is a Boolean resource that controls whether *xdm* generates and uses authorization for the local server connections. If authorization is used, `authName` is a whitespace-separated list of authorization mechanisms to use. XDMCP connections dynamically specify which authorization types are supported, so `authName` is ignored in this case. When `authorize` is set for a display and authorization is not available, the user is informed by having a different message displayed in the login widget. By default, `authorize` is `True` and `authName` is MIT-MAGIC-COOKIE-1.

`DisplayManager.`*`DISPLAY`*`.authFile`

This file is used to communicate the authorization data from *xdm* to the server, using the `-auth` server command-line option. It should be kept in a directory which is not world-writeable, as it could easily be removed, disabling the authorization mechanism in the server.

`DisplayManager.DISPLAY.authComplain`

If set to `False`, disables the use of the `unsecureGreeting` in the login window. See the section "Authentication Widget." The default is `True`. (Available as of Release 5.)

`DisplayManager.DISPLAY.resetSignal`

The number of the signal *xdm* sends to reset the server. See the section "Controlling the Server." The default is 1 (SIGHUP). (Available as of Release 5.)

`DisplayManager.DISPLAY.termSignal`

The number of the signal *xdm* sends to terminate the server. See the section "Controlling the Server." The default is 15 (SIGTERM). (Available as of Release 5.)

`DisplayManager.`*`DISPLAY`*`.resetForAuth`

The original implementation of authorization in the sample server rereads the authorization file at server reset time, instead of when checking the initial connection. As *xdm* generates the authorization information just before connecting to the display, an

old server would not get up-to-date authorization information. This resource causes *xdm* to send SIGHUP to the server after setting up the file, causing an additional server reset to occur, during which time the new authorization information will be read. The default is `false`, which will work for all MIT servers.

`DisplayManager.`*`DISPLAY`*`.userAuthDir`
When *xdm* is unable to write to the usual user authorization file (*$HOME/.Xauthority*), it creates a unique filename in this directory and points the environment variable XAUTHORITY at the created file. By default, it uses */tmp*.

XDMCP Access Control

The database file specified by the `DisplayManager.accessFile` provides information that *xdm* uses to control access from displays requesting XDMCP service. This file contains three types of entries: entries which control the response to Direct and Broadcast queries, entries which control the response to Indirect queries, and macro definitions.

The format of the Direct entries is simple, either a hostname or a pattern. A pattern is distinguished from a hostname by the inclusion of one or more meta characters. ("*" matches any sequence of 0 or more characters, and "?" matches any single character.) The pattern is then compared against the hostname of the display device. If the entry is a hostname, all comparisons are done using network addresses, so any name which converts to the correct network address may be used. For patterns, only canonical hostnames are used in the comparison, so ensure that you do not attempt to match aliases. Preceding either a hostname or a pattern with a "!" character causes hosts that match that entry to be excluded.

An Indirect entry also contains a hostname or pattern, but follows it with a list of hostnames or macros to which indirect queries should be sent.

A macro definition contains a macro name and a list of hostnames and other macros that the macro expands to. To distinguish macros from hostnames, macro names start with a "%" character. Macros may be nested.

Indirect entries may also specify to have *xdm* run *chooser* to offer a menu of hosts to connect to. See the section "Chooser."

When checking access for a particular display host, each entry is scanned in turn and the first matching entry determines the response. Direct and Broadcast entries are ignored when scanning for an Indirect entry and vice-versa.

Blank lines are ignored; "#" is treated as a comment delimiter causing the rest of that line to be ignored, and "*newline*" causes the newline to be ignored, allowing indirect host lists to span multiple lines.

Here is an example Xaccess file:

```
#
# Xaccess - XDMCP access control file
#

#
```

```
# Direct/Broadcast query entries
#
!xtra.lcs.mit.edu          # disallow direct/broadcast service for xtra
bambi.ogi.edu              # allow access from this particular display
*.lcs.mit.edu              # allow access from any display in LCS

#
# Indirect query entries
#
%HOSTS                     expo.lcs.mit.edu xenon.lcs.mit.edu \\e
                           excess.lcs.mit.edu kanga.lcs.mit.edu

extract.lcs.mit.edu        xenon.lcs.mit.edu  #force extract to contact xenon
!xtra.lcs.mit.edu          dummy              #disallow indirect access
*.lcs.mit.edu              %HOSTS             #all others get to choose
```

Chooser

For X terminals that do not offer a host menu for use with Broadcast or Indirect queries, the *chooser* program can do this for them. (The *chooser* is available as of Release 5.) In the *Xaccess* file (another Release 5 innovation), specify CHOOSER as the first entry in the Indirect host list. *chooser* will send a Query request to each of the remaining hostnames in the list and offer a menu of all the hosts that respond.

The list may consist of the word BROADCAST, in which case *chooser* will send a Broadcast instead, again offering a menu of all hosts that respond. Note that on some operating systems, UDP packets cannot be broadcast, so this feature will not work.

Here's an example *Xaccess* file using *chooser*:

```
extract.lcs.mit.eduCHOOSER %HOSTS#offer a menu of these hosts
xtra.lcs.mit.eduCHOOSER BROADCAST#offer a menu of all hosts
```

The program to use for *chooser* is specified by the `DisplayManager.DISPLAY.chooser` resource. Resources for this program can be put into the file named by `DisplayManager.DISPLAY.resources`.

Server Specification

The resource `DisplayManager.servers` gives a server specification, or, if the values start with a slash (/), the name of a file containing server specifications, one per line.

Each specification indicates a display which should constantly be managed, and which is not using XDMCP. Each consists of at least three parts: a display name, a display class, a display type, and (for local servers) a command line to start the server. A typical entry for local display number 0 would be:

```
:0 Digital-QV local /usr/bin/X11/X :0
```

The display types are:

local	Local display: *xdm* must run the server
foreign	Remote display: *xdm* opens an X connection to a running server

The display name must be something that can be passed in the `-display` option to an X program. This string is used to generate the display-specific resource names, so be careful to match the names (e.g., use `:0 local /usr/bin/X11/X :0` instead of `localhost:0 local /usr/bin/X11/X :0` if your other resources are specified as `DisplayManager._0.session`). The display class portion is also used in the display-specific resources, as the class of the resource. This is useful if you have a large collection of similar displays (like a corral of X terminals) and would like to set resources for groups of them. When using XDMCP, the display is required to specify the display class, so the manual for your particular X terminal should document the display class string for your device. If it doesn't, you can run *xdm* in debug mode and look at the resource strings which it generates for that device, which will include the class string.

Setup Program

The *Xsetup* file is run after the server is reset, but before the Login window is offered. The file is typically a shell script. It is run as root, so it should be careful about security. This is the place to change the root background or bring up other windows that should appear on the screen along with the Login widget.

In addition to any specified by `DisplayManager.exportList`, the following environment variables are passed:

DISPLAY	The associated display name
PATH	The value of `DisplayManager.DISPLAY.systemPath`
SHELL	The value of `DisplayManager.DISPLAY.systemShell`
XAUTHORITY	May be set to an authority file

Note that since *xdm* grabs the keyboard, any other windows will not be able to receive keyboard input. They will be able to interact with the mouse; however, beware of potential security holes here. If `DisplayManager.DISPLAY.grabServer` is set, *Xsetup* will not be able to connect to the display at all. Resources for this program can be put into the file named by `DisplayManager.DISPLAY.resources`.

Authentication Widget Resources

The authentication widget reads a name/password pair from the keyboard. Nearly every imaginable parameter can be controlled with a resource. Resources for this widget should be put into the file named by `DisplayManager.DISPLAY.resources`. All of these resources have reasonable default values, so it is not necessary to specify any of them.

`xlogin.Login.width`, `xlogin.Login.height`, `xlogin.Login.x`, `xlogin.Login.y`
: The geometry of the Login widget is normally computed automatically. If you wish to position it elsewhere, specify each of these resources.

`xlogin.Login.foreground`
: The color used to display the typed-in user name.

`xlogin.Login.font`
: The font used to display the typed-in user name.

`xlogin.Login.greeting`
: A string which identifies this window. The default is "X Window System".

`xlogin.Login.unsecureGreeting`
: When X authorization is requested in the configuration file for this display and none is in use, this greeting replaces the standard greeting. The default is "This is an unsecure session".

`xlogin.Login.greetFont`
: The font used to display the greeting.

`xlogin.Login.greetColor`
: The color used to display the greeting.

`xlogin.Login.namePrompt`
: The string displayed to prompt for a user name. *xrdb* strips trailing white space from resource values, so to add spaces at the end of the prompt (usually a nice thing), add spaces escaped with backslashes. The default is "Login:".

`xlogin.Login.passwdPrompt`
: The string displayed to prompt for a password. The default is "Password:".

`xlogin.Login.promptFont`
: The font used to display both prompts.

`xlogin.Login.promptColor`
: The color used to display both prompts.

`xlogin.Login.fail`
: A message which is displayed when the authentication fails. The default is "Login incorrect".

`xlogin.Login.failFont`
: The font used to display the failure message.

`xlogin.Login.failColor`
: The color used to display the failure message.

`xlogin.Login.failTimeout`
: The number of seconds that the fail message is displayed. The default is 30.

`xlogin.Login.translations`
: This specifies the translations used for the login widget. See Chapter 11, *Setting Resources*, and Appendix F, *Translation Table Syntax*, for more information on translations. The default translation table for *xdm* is:

```
Ctrl<Key>H:         delete-previous-character() \n\
Ctrl<Key>D:         delete-character() \n\
Ctrl<Key>B:         move-backward-character() \n\
Ctrl<Key>F:         move-forward-character() \n\
Ctrl<Key>A:         move-to-beginning() \n\
Ctrl<Key>E:         move-to-end() \n\
Ctrl<Key>K:         erase-to-end-of-line() \n\
Ctrl<Key>U:         erase-line() \n\
Ctrl<Key>X:         erase-line() \n\
Ctrl<Key>C:         restart-session() \n\
Ctrl<Key>\\:        abort-session() \n\
<Key>BackSpace:     delete-previous-character() \n\
<Key>Delete:        delete-previous-character() \n\
<Key>Return:        finish-field() \n\
<Key>:              insert-char() \
```

The actions that are supported by the widget are:

delete-previous-character
: Erases the character before the cursor.

delete-character
: Erases the character after the cursor.

move-backward-character
: Moves the cursor backward one character.

move-forward-character
: Moves the cursor forward one character.

move-to-beginning
: Moves the cursor to the beginning of the editable text.

move-to-end
: Moves the cursor to the end of the editable text.

erase-to-end-of-line
: Erases all text after the cursor.

erase-line
: Erases the entire text.

finish-field
: If the cursor is in the `name` field, proceeds to the `password` field; if the cursor is in the `password` field, checks the current name/password pair. If the name/password pair is valid, *xdm* starts the session. Otherwise, the failure message is displayed and the user is prompted again.

abort-session
: Terminates and restarts the server.

abort-display
: Terminates the server, disabling it. This is a rash action and is not accessible in the default configuration. It can be used to stop *xdm* when shutting the system down or when using *xdmshell*.

restart-session
: Resets the X server and starts a new session. This can be used when the resources have been changed and you want to test them, or when the screen has been overwritten with system messages.

insert-char
: Inserts the character typed.

set-session-argument
: Specifies a single word argument which is passed to the session at startup. See the sections "The Xsession Program" and "Typical Usage."

allow-all-access
: Disables access control in the server. This can be used when the *.Xauthority* file cannot be created by *xdm*. Be very careful when using this; it might be better to disconnect the machine from the network first.

The Startup Program

The *Xstartup* file is typically a shell script. It is run as root, and so you should be very careful about security. This is the place to put commands that add entries to */etc/utmp*, mount users' home directories from file servers, display the message of the day, or abort the session if logins are not allowed. Various environment variables are set for the use of this script:

DISPLAY	The associated display name.
HOME	The home directory of the user.
USER	The user name.
PATH	The value of `DisplayManager.DISPLAY.systemPath`.
SHELL	The value of `DisplayManager.DISPLAY.systemShell`.
XAUTHORITY	May be set to an authority file.

No arguments are passed to the script. *xdm* waits until this script exits before starting the user session. If the exit value of this script is non-zero, *xdm* discontinues the session and starts another authentication cycle.

The Session Program

The *Xsession* file is the script that is run as the user's session. It is run with the permissions of the authorized user and has several environment variables specified:

DISPLAY	The associated display name.
HOME	The home directory of the user.
USER	The user name.
PATH	The value of `DisplayManager.DISPLAY.userPath`.
SHELL	The user's default shell (from *getpwnam*).
XAUTHORITY	May be set to a nonstandard authority file.

At most installations, *Xsession* should look in $HOME for a file *.xsession*, which contains commands that each user would like to use as a session. *Xsession* should also implement a system default session if no user-specified session exists. See "Typical Usage."

An argument may be passed to this program from the authentication widget using the "set-session-argument" action. This can be used to select different styles of session. One very good use of this feature is to allow the user to escape from the ordinary session when it fails. This would allow users to repair their own *.xsession* if it fails, without requiring administrative intervention. The section "Typical Usage" demonstrates this feature.

The Reset Program

Symmetrical with *Xstartup*, the *Xreset* script is run after the user session has terminated. Run as root, it should contain commands that undo the effects of commands in *Xstartup*, removing entries from */etc/utmp* or unmounting directories from file servers. The environment variables that were passed to *Xstartup* are also passed to *Xreset*.

Typical Usage

Actually, *xdm* is designed to operate in such a wide variety of environments that "typical" is probably a misnomer. First, the *xdm* configuration file should be set up. Make a directory (commonly */usr/lib/X11/xdm*) to contain all of the relevant files. Here is a reasonable configuration file for Release 5, which could be named *xdm-config*:

```
DisplayManager.servers:                 /usr/lib/X11/xdm/Xservers
DisplayManager.errorLogFile:            /usr/lib/X11/xdm/xdm-errors
DisplayManager*resources:               /usr/lib/X11/xdm/Xresources
DisplayManager*startup:                 /usr/lib/X11/xdm/Xstartup
DisplayManager*session:                 /usr/lib/X11/xdm/Xsession
DisplayManager.pidFile:                 /usr/lib/X11/xdm/xdm-pid
DisplayManager._0.authorize:            true
DisplayManager*authorize:               false
```

As you can see, the *xdm-config* file primarily contains references to other files. Note that some of the resources are specified with an asterisk (*) separating the components. These resources can be made unique for each different display by replacing the "*" with the display name, but normally this is not very useful. See the "Resources" section for a complete discussion.

The first file, */usr/lib/X11/xdm/Xservers*, contains the list of displays to manage that are not using XDMCP. Most workstations have only one display, numbered 0, so the file will look something like this:

```
:0 Local local /usr/bin/X11/X :0
```

This will keep */usr/bin/X11/X* running on this display and manage a continuous cycle of sessions.

The file */usr/lib/X11/xdm/xdm-errors* will contain error messages from *xdm* and anything output to standard error by *Xsetup*, *Xstartup*, *Xsession*, or *Xreset*. When you have trouble getting *xdm* working, check this file to see if *xdm* has any clues to the trouble.

The next configuration entry, */usr/lib/X11/xdm/Xresources*, is loaded onto the display as a resource database using *xrdb*. As the authentication widget reads this database before starting up, it usually contains parameters for that widget:

```
xlogin*login.translations: #override\\e
   <Key>F1: set-session-argument(failsafe) finish-field()\\en\\e
   <Key>Return: set-session-argument() finish-field()
xlogin*borderWidth: 3
#ifdef COLOR
xlogin*greetColor:  CadetBlue
xlogin*failColor:   red
#endif
```

Please note the translations entry; it specifies a few new translations for the widget, which allows users to escape from the default session (and avoid troubles that may occur in it). Note that if `#override` is not specified, the default translations are removed and replaced by the new value, not a very useful result, as some of the default translations are quite useful (such as `<Key>: insert-char()`, which responds to normal typing!).

The *Xstartup* file used here simply prevents login while the file */etc/nologin* exists. As there is no provision for displaying any messages here (there isn't any core X client which displays files), the user will probably be baffled by this behavior. The following sample *Xstartup* file is not a complete example, simply a demonstration of the available functionality:

```
#!/bin/sh
#
# Xstartup
#
# This program is run as root after the user is verified
#
if [ -f /etc/nologin ]; then
        exit 1
```