The Definitive Guides to the X Window System

Volume Three

Motif Edition

X Window System User's Guide

OSF/Motif 1.2 Edition

By Valerie Quercia and Tim O'Reilly

O'Reilly & Associates, Inc.

WINDOW SYSTEM

O'Reilly & Associates, Inc.

X Window System User's Guide

Motif

3

Edition

0002

X Window System User's Guide

Volume Three

X Window System User's Guide

OSF/Motif 1.2 Edition

by Valerie Quercia and Tim O'Reilly

O'Reilly & Associates, Inc.

X Window System User's Guide

by Valerie Quercia and Tim O'Reilly

Editor: Tim O'Reilly

Printing History:

September 1988:	Athena First Edition.
July 1989:	Athena Second Edition. Revised to reflect Release 3.
October 1989:	Athena Edition. Minor corrections.
May 1990:	Athena Third Edition. Revised to reflect Release 4.
January 1991:	Motif First Edition.
March 1992:	Motif Edition. Minor corrections.
January 1993:	Motif Second Edition. Revised to reflect Release 5 and Motif 1.2.
May 1993:	Motif Edition. Minor corrections.
August 1993:	Motif Edition. Minor corrections.

Motif Edition ISBN 1-56592-015-5 [5/98]

Table of Contents

Figures

Examples

Tables

Preface

The preface provides a map through the most important topics covered in this guide. It also describes the book's conventions.

In the Preface:

Preface

The X Window System is a network-based graphics windowing system. It was developed by MIT and has been adopted as an industry standard. X provides the bare bones of a window system upon which almost any style of graphical user interface (GUI) can be built. One of the most popular user interfaces available in the market today is OSF/Motif developed by the Open Software Foundation.

The *X Window System User's Guide, Motif Edition* describes window system concepts, the application programs (clients) commonly distributed with Version 11, Release 5 of X, and how you can expect programs to operate with OSF/Motif. (The *Motif Edition* is intended for those using X with the OSF/Motif interface. Another edition of the *X Window System User's Guide* is available for those using X without Motif.)

Do I Have To Read This WHOLE Book?

Let's end the suspense: the answer is no. Though the sheer weight of this guide implies that learning X is a difficult task, this is not strictly true. You can quickly learn enough about the X Window System to begin working productively.

A Quick Start Approach to X

Much of this guide (all of Part Three, for instance) is reference material, which we think you'll want to have after working a while, but which you certainly don't have to digest to get going.

The current section outlines what you probably need to know to begin working productively and gives suggestions on shortcuts through some of the other material. Throughout the guide, we've also tried to let you know how to get to the most basic material. (Pay attention to the introductory section of each chapter.)

Once you know your way around the X environment, you may want to take advantage of additional clients and customization features. The remainder of the guide should help you.

For starters, we think you should read the following sections:

Chapter 1, *An Introduction to the X Window System*
: Read all of "Anatomy of an X Display" and "Standard X Clients versus Motif Clients." Browse the subsequent sections about "System Architecture" and get an idea what the basic "Clients" (i.e., programs) are. The sections "Other Standard X Clients" and "Customizing Clients" will help you figure out what other system features you might eventually want to take advantage of.

Chapter 2, *Getting Started*
: The already short Chapter 2 is written to direct you through it in the briefest possible time. Follow the pointers in the chapter. The information you need will become apparent.

Chapter 3, *Working in the X Environment*
: If you read *anything* in this guide, make it Chapter 3. It's a little on the long side, but it covers virtually everything you need to begin working productively. If you get bogged down, think about the chapter covering roughly three topics: window management, setting up your display, and some rudimentary client customization.

Chapter 5, *The xterm Terminal Emulator*
: A terminal emulator is probably the most important program you will use. Read the following sections (and any subsections): "Running xterm with a Scrollbar," "Copying and Pasting Text Selections," and the "VT Fonts Menu."

That's it! About 70 pages—many of which have illustrations. :-) The remaining chapters deal with the range of available clients and how to customize them. Now, you may wonder, if it's really so easy, why do you need so much information? In other words, why is the *X Window System User's Guide* so long? The answer is also simple; because we think you'll soon outgrow the basic skills and want to know more.

The most basic X session requires only two programs (called *clients* in X parlance): a terminal emulator, a window which provides all the functionality of a standard terminal; and a window manager, which allows you to move, resize, and otherwise control the windows on your screen. You should have a working knowledge of these programs after getting through the "Quick Start Approach" we've outlined. Running even just these two applications should help you appreciate the benefits of a windowing system such as X.

However, even the most basic incarnation of the X Window System includes dozens of other useful utilities. We provide tutorials for many of these clients in Part One (and a few in Part Two) of this guide. Read "Other Standard X Clients" at the end of Chapter 1 and browse through the following chapters to get a better idea of what sorts of programs are available: Chapter 7, *Graphics Utilities*, Chapter 8, *Other Clients*, and Chapter 14, *Setup Clients*.

In addition, as you'll soon see, you can customize virtually every feature of X to better suit your own needs. For instance, once you get used to moving the pointer around and see how the cursor symbol follows it on the screen, you might want to specify a different cursor. You can do that using the *xsetroot* client described in Chapter 14. Appendix D, *Standard Cursors* lists some of the alternative symbols you can use. If you're really ambitious, you can even create your own symbol using the *bitmap* client, described in Chapter 7, *Graphics Utilities*. (Changing a cursor symbol is just one of the thousands of customizations you can make within the X environment.)

Even the most experienced user will probably need to verify command-line options and so-called resource variables that can be used to customize client appearance and behavior. Part Two of the *X Window System User's Guide, Motif Edition* describes how to customize X, and the reference pages in Part Three detail the valid command-line options and resource variables for each of the standard clients.

Additional reference information appears in Part Four, the Appendices. This information includes the names and pictures of some of the fonts, bitmaps, and cursor symbols available with X. These graphic aids should be helpful in designing your user environment. Other appendices present more "technical" information, including some rudimentary X system management.

Before we let you get started, here are some additional maps through the material. The following sections deal with some potentially important topics:

Chapter 4, *More about the mwm Window Manager*
: The window management skills taught in Chapter 3 will probably be enough to get you started. If you run up against a problem you can't solve, the sections "Using the Window Menu" and "The Root Menu" might provide the answer.

Chapter 6, *Font Specification*
: The section "If You Just Want to Pick a Font" simplifies choosing and specifying a readable X font.

Chapter 10, *Command-line Options*
: A short chapter outlining some powerful "standard" options that can be used with most clients.

Chapter 11, *Setting Resources*
: You can get a basic idea of how to customize clients using resource variables by reading the following sections of Chapter 11: "Resource Naming Syntax" and "How to Set Resources."

Chapter 12, *Specifying Color*
: The section "Available RGB Colors" (and its subsections) survey the color names that you should be able to specify on any X system.

Appendix A, *Managing Your Environment*
: "A Startup Shell Script" can automatically bring up the environment you want.

You can learn more about the Motif window manager and Motif Toolkit applications by reading Chapter 13, *Customizing mwm*, and Chapter 9, *Working with Motif Applications*, respectively.

We hope these guidelines will help you get working quickly—and maybe experimenting with the flexible X Window System. There are many more pointers throughout the text. Best of luck!

Assumptions

This book assumes that X is already installed on your system and that all standard MIT clients are available. In addition, although X runs on many different systems, this book assumes that you are running it on a UNIX system and that you have basic familiarity with UNIX. If you are not using UNIX, you will still find this book useful—UNIX dependencies are not that widespread—but you may occasionally need to translate a command example into its equivalent on your system. This book also assumes that you are using a three-button pointer (e.g., a mouse) and that the operation of the *mwm* window manager is controlled by the *system.mwmrc* file provided with OSF/Motif 1.2. (If this is not the case, the book provides information that will allow you to understand how *mwm* is configured on your system.)

This book is written for both first-time and experienced users of the X Window System. First-time users should use the preceding section, "A Quick Start Approach to X," as a guide to getting started.

Experienced users can use this book as a reference for the client programs detailed here. Since there is great flexibility with X, even frequent users need to check on the syntax and availability of options. Reference pages for each client detail command-line options, customization database (resource database) variables, and other detailed information.

What's New in Release 5?

This guide describes Release 5 of X from MIT as it runs with the OSF/Motif graphical user interface. Throughout the guide, we've tried to highlight what's new. (Look for pointers in the text.) The current section gives an overview of the Release 5 innovations that may be relevant to users and also directs you to further information.

New Client Programs

The following programs (*clients* in X parlance) have been added to X at R5:

bdftopcf	Converts a font file from bitmap distribution format to portable compiled format.
fs	A font server program. (See Chapter 6, *Font Specification*, for more information.)
fsinfo	Provides information about a font available from a font server. (See Chapter 6.)
fstobdf	Converts a font file from font server format to bitmap distribution format.
fslsfonts	Lists fonts available from a font server. (See Chapter 6.)
showfont	Displays data about a font available from a font server. See Chapter 6 and Chapter 7.

editres	Helps you set resource variables that control how programs look and operate. (See Chapter 11, *Setting Resources*.)
viewres	Displays a tree showing the organization of a program's component parts (known as widgets).
xcmsdb	Helps you manage color screen data.
xconsole	Intercepts system console messages that would otherwise obscure your screen. (See Appendix A, *Managing Your Environment*.)

Each of these programs is also described in a reference page in Part Three of this guide.

Defunct Programs

The following clients have been removed from the standard distribution in Release 5:

bdftosnf	Converts a font file from bitmap distribution format to server natural format. *bdftosnf* has in effect been replaced by the *bdftopcf* program.
xlswins	Shows how windows on the display are related to one another. *xlswins* has been replaced by the `-tree` and `-children` options to *xwininfo*.

Clients that Work Differently in Release 4 and Release 5

Most clients vary slightly from one release to the next, though in many cases the differences are superficial (e.g., the names of menu items might change) or not immediately obvious (e.g., new options may be available). The following clients look or function significantly differently in Release 4 and Release 5.

bitmap	The *bitmap* editor has been radically redesigned in Release 5. Notable improvements include copy and paste capabilties and dynamic file loading. Chapter 7, *Graphics Utilities*, provides an overview of what's new and also extensive tutorials.
xmag	The *xmag* magnification program has been redesigned to work in concert with *bitmap*. You can now copy and paste between the two applications. See Chapter 7.

R5 changes for other clients are detailed in the tutorials throughout the guide and should also be noted on the reference pages in Part Three.

Available Fonts

Release 5 adds several miscellaneous fonts, including two Hangul fonts, two Hebrew fonts, an additional Kanji font. R5 also sees the addition of *font scaling* and includes some new *outline fonts*, which are suitable for scaling.

The new font server program (*fs*) provides access to fonts resident on other machines in the network.

Chapter 6, *Font Specification*, describes X's latest font capabilities.

Available Colors

Predefined colors and their text names appear in the file *rgb.txt*, which is generally located in */usr/lib/X11*. These so-called RGB colors are not portable—that is, they look different on different monitors.

In Release 5, X also provides *device-independent* color capabilities. Now you can specify colors that will look the same, regardless of the type of screen. See Chapter 12 for details.

Screen-specific Resources

R5 allows you to specify different application defaults for color and monochrome screens. Use the mechanisms described in Chapter 11 to specify program characteristics that will be automatically implemented depending on the type of monitor you're using.

Bulk Sales Information

This guide is being resold by many workstation manufacturers as their official X Window System documentation. For information on volume discounts for bulk purchases, call O'Reilly & Associates, Inc., at 617-354-5800, or send email to linda@ora.com (uunet!ora!linda). For companies requiring extensive customization of the guide, source-licensing terms are also available.

Acknowledgements

The current edition of this guide was written by Valerie Quercia, with the invaluable editorial assistance of Tim O'Reilly. Special thanks to Linda Mui and Eric Pearce (the co-authors of Volume Eight, *X Window System Administrator's Guide*) for their expert technical and editorial support. Thanks also to Paula Ferguson for updating Appendix G, *Widget Resources*.

Previous editions of this guide were written by Valerie Quercia and Tim O'Reilly, with the exception of the first edition, which was written by Tim O'Reilly, Valerie Quercia, and Linda Lamb. The first edition was based in part on three previous X Window System user's guides: one from Masscomp, which was written by Jeff Graber; one from Sequent Computer Systems, Inc., and one from Graphic Software Systems, Inc.; both of which were written by Candis Condo (supported by the UNIX development group). Some of Jeff's and Candis's material in turn was based on material developed under the auspices of Project Athena at MIT.

Most of the reference pages in Part Three of this guide have been adapted from reference pages copyright © 1991 the Massachusetts Institute of Technology, or from reference pages produced by Graphic Software Systems. Refer to the "Authors" section at the end of each reference page for details. Other copyrights are listed on the relevant reference pages.

Permission to use this material is gratefully acknowledged.

This guide was primarily developed using the MIT sample server on a Sun-3 Workstation and a Sun SPARCstation™ SLC, with additional testing done on a Sony NEWS workstation running Sony's X implementation, a Visual 640 X Display Station, and NCD16™ and NCD19 Network Display Stations.

We'd also like to thank the Open Software Foundation for permission to reprint the *system.mwmrc* file in Chapter 13, *Customizing mwm*. Special thanks to Elizabeth Connelly of OSF for arranging this.

Of course, the authors would also like to thank those who worked on the production of the book: Donna Woonteiler and Leslie Chalmers for copy editing, Chris Reilly and Jeff Robbins for creating the illustrations, Ellie Cutler for indexing, and Leslie Chalmers, Kismet McDonough, Laura Parker, and Mike Sierra for helping to produce the final copy. Thanks also to Lenny Muellner for his expert tools assistance.

Despite the efforts of these people, the standard authors' disclaimer applies: any errors that remain are our own.

Font and Character Conventions

These typographic conventions are used in this book:

Italics	are used for: • new terms where they are defined. • file and directory names, and command and client names when they appear in the body of a paragraph.
`Courier`	is used within the body of the text to show: • command lines or options that should be typed verbatim on the screen. is used within examples to show: • computer-generated output. • the contents of files.
`Courier bold`	is used within examples to show command lines and options that should be typed verbatim on the screen.
`Courier italics`	are used within examples or explanations of command syntax to show a parameter to a command that requires context-dependent substitution (such as a variable). For example, *`filename`* means to use some appropriate filename; *`option(s)`* means to use some appropriate option(s) to the command.
Helvetica	is used to show menu titles and options.

These symbols are used within the *X Window System User's Guide*:

[ ]	surround an optional field in a command line or file entry.
$	is the standard prompt from the Bourne shell, *sh*(1).
%	is the standard prompt from the C shell, *csh*(1).
name(1)	is a reference to a command called *name* in Section 1 of the *UNIX Reference Manual* (which may have a different name depending on the version of UNIX you use).

Part One:

Using X

Part One provides an overview of the X Window System and concepts, and describes how to use the most important programs available in the X environment.

1

An Introduction to the X Window System

This chapter describes the features of a typical X display, while introducing some basic window system concepts. It also provides an overview of the X Window System's client-server architecture and briefly describes the most commonly used clients.

In This Chapter:

1
An Introduction to the X Window System

The X Window System, called X for short, is a network-based graphics window system that was developed at MIT in 1984. Several versions of X have been developed, the most recent of which is X Version 11 (X11), first released in 1987.*

X11 has been adopted as an industry standard windowing system. X is supported by a consortium of industry leaders such as DEC, Hewlett-Packard, Sun, IBM, and AT&T that have united to direct, contribute to, and fund its continuing development. In addition to the system software development directed by the X Consortium, many independent developers are producing application software specifically for use with X, including spreadsheets, database programs, and publishing applications.

Before you plunge into Chapter 1, we suggest you read "A Quick Start Approach to X" in the Preface. This section will explain how to slice your way through the guide to some of the most basic and pertinent information.

Then we'll take a look at a typical X display and consider some general system features. We'll also briefly compare a standard X application (written with the X Toolkit) to a Motif application (written with the Motif Toolkit). Then we'll discuss what distinguishes the X Window System from other window systems. We'll also introduce some of the more important programs included in the standard distribution of X, and the *mwm* window manager shipped with OSF/Motif.

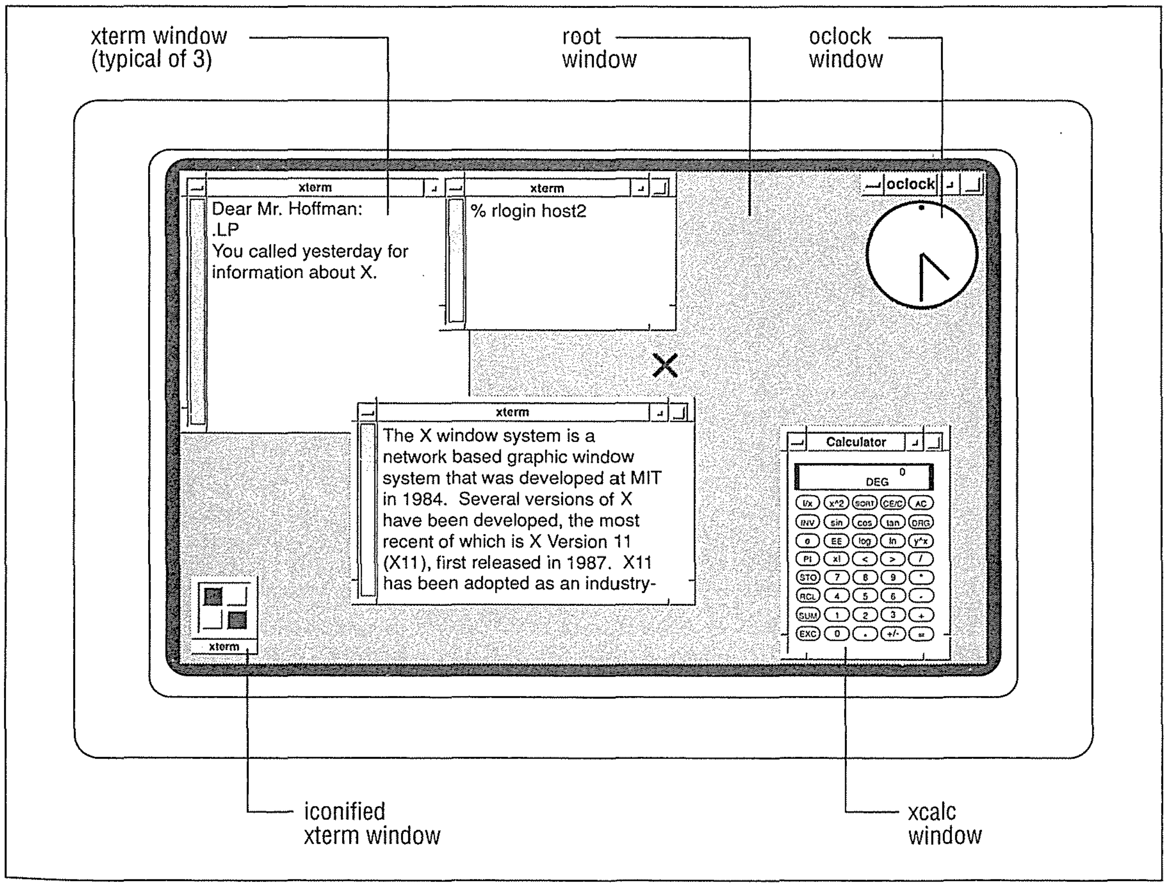
*There have been several revisions to X since then. This guide describes the latest release, X11 R5.

Anatomy of an X Display

X is typically run on a workstation with a large screen or on a special graphics terminal known as an X terminal. (It also runs on PCs and on many larger systems.) X allows you to work with multiple programs simultaneously, each in a separate *window*. The display in Figure 1-1 includes five windows.

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

The windows you will probably use most frequently are *terminal emulators*, windows that function as standard terminals. The terminal emulator included with the standard release of X is called *xterm*. Figure 1-1 depicts three *xterm* windows. In an *xterm* window, you can do anything you might do in a regular terminal: enter commands, run editing sessions, compile programs, etc.



Dear Mr. Hoffman:
.LP
You called yesterday for information about X.

% rlogin host2

The X window system is a network based graphic window system that was developed at MIT in 1984. Several versions of X have been developed, the most recent of which is X Version 11 (X11), first released in 1987. X11 has been adopted as an industry-

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

The display in Figure 1-1 also includes two other application windows: a clock (called *oclock*) and a calculator (*xcalc*). X provides many such small utility programs—analogous to the so-called "desk accessories" of the Macintosh environment—intended to make your work easier.

The shaded area that fills the entire screen is called the *root* (or *background*) window. Application windows are displayed on top of this root window. X considers windows to be related to one another in a hierarchy, similar to a family tree. The root window is the root or origin within this hierarchy and is considered to be the *parent* of application windows displayed on it. Conversely, these application windows are called *children* of the root window. In Figure 1-1, the *xterm*, *oclock*, and *xcalc* windows are children of the root window.

As we'll see later, the window hierarchy is actually much more complicated than this "two generation" model suggests. Various *parts* of application windows are windows in their own right. For example, many applications provide menus to assist the user. Technically speaking, these menus are separate windows. Knowledge of the complexity of the window hierarchy (and the composite parts of an application) will become important when we discuss how to tailor an application to better suit your needs.

One of the strengths of a window system such as X is that you can have several processes going on simultaneously in several different windows. For example, in Figure 1-1, the user is logging in to a remote system in one *xterm* window and editing a text file in each of the two other *xterm* windows. (As we'll see in Chapter 5, *The xterm Terminal Emulator*, you can also cut and paste text between two windows.) Be aware, however, that you can only input to one window at a time.

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

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

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

The window manager provided with OSF/Motif is called *mwm*, the *M*otif *w*indow *m*anager. The most distinguishing feature of *mwm* is the "frame" it places around all windows on the display. Notice that most top-level application windows on our typical display are surrounded by this frame. (The *oclock* window is an exception. With non-rectangular windows such as this, a titlebar appears to "float" above the window.) As we'll see, by clicking a mouse or other pointing device on various parts of the window frame (or titlebar, if that's what's available), you can perform management functions on the window.

The *mwm* window frame is a composite of several parts, the most prominent of which are shown in Figure 1-2.

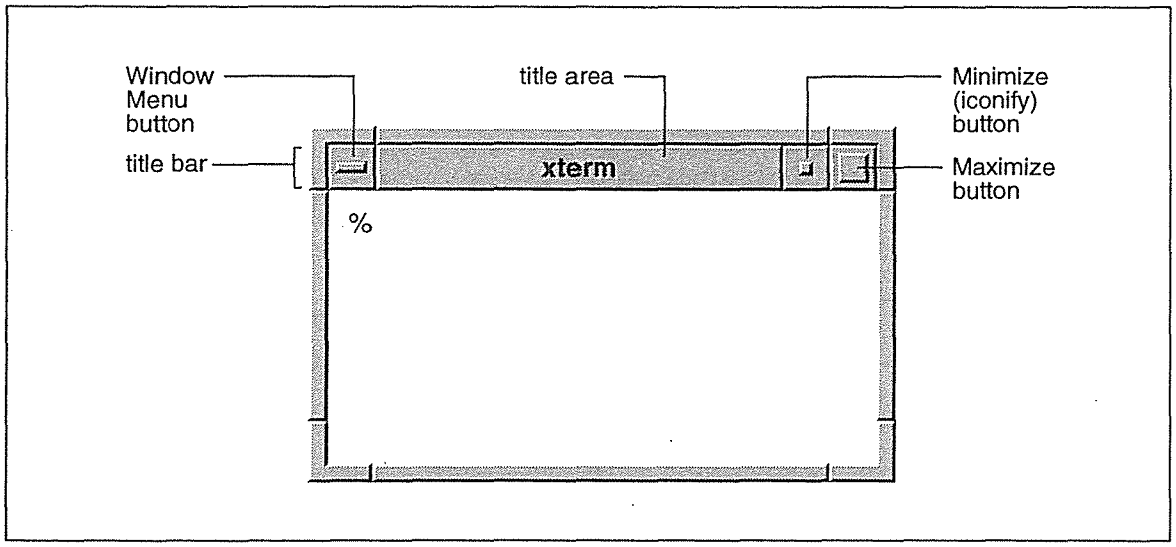


Figure 1-2. mwm frames each window on the display

The top edge of the frame is wider than the other three edges and features most of the window management tools. This wide horizontal bar spanning the top of the window is known as a *titlebar*. The large central portion of this top edge is called the *title area* mainly because it contains a text description of the window. (Generally, this is the application name, but as we'll see later, you can often specify an alternate title.) The titlebar also features three command buttons whose functionality we'll discuss in Chapter 3 and Chapter 4. We'll also see how to use the sides and bottom of the frame to resize a window and to raise it to the top of the window stack.

mwm attempts to create a three-dimensional appearance, which is somewhat more aesthetic than the look provided by many other window managers. You'll probably notice that window frames, various command buttons, icons, etc., appear to be raised to varying heights above screen level. This illusion is created by subtle shading and gives many display features a "beveled" look, similar to the beveled style of some mirrors.

mwm is intended to be used with the OSF/Motif graphical user interface. For those not using Motif, there are several other window managers available in the market today. In the standard distribution of X from MIT, the official window manager is called *twm*. (*twm* originally stood for "Tom's window manager," in honor of its developer, Tom LaStrange. However, it has since been renamed the "tab window manager.") The *twm* window manager provides a different "look and feel" than *mwm*. Rather than framing application windows, *twm* simply provides each window with a titlebar, different from the *mwm* titlebar in style, but offering similar window management functions. The user-contributed part of the MIT X distribution includes several other window managers and still others are available commercially.

Aesthetics notwithstanding, one of the primary advantages *mwm* has over other window managers is inherent in the nature of a frame: it provides window management tools on four sides of the window. *twm*'s titlebar is a useful window management tool, but a titlebar is often covered by other windows in the stack. In most cases at least a part of a window's frame should be visible—and thus accessible to the user.

Also pictured in Figure 1-1 is an *icon*. An icon is a small symbol that represents a window in an inactive state. The window manager allows you to convert windows to icons and icons back to windows. You may want to convert a window to an icon to save space on the display or to prevent input to that window. Each icon has a label, generally the name of the program that created the window. The icon in Figure 1-1 represents a fourth *xterm* window on the display. Icons can be moved around on the display, just like active windows.

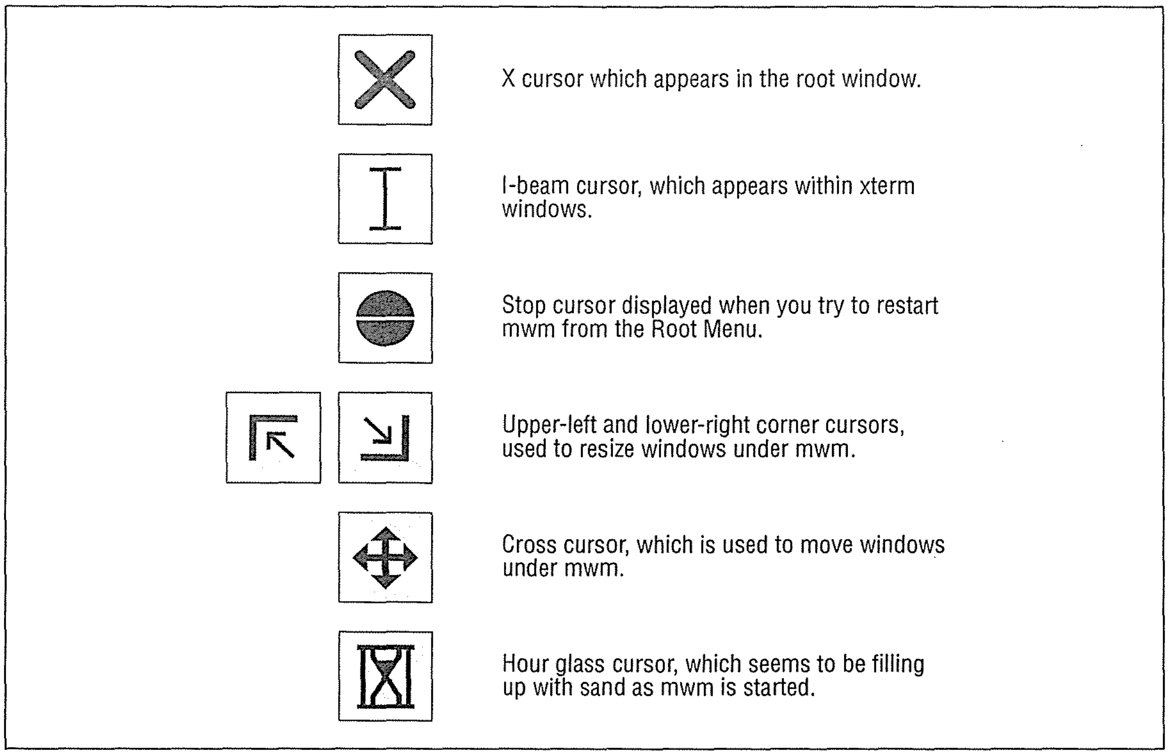
The contents of a window are not visible when the window has been converted to an icon but they are not lost. In fact, a client continues to run when its window has been iconified; if you iconify a terminal emulator client such as *xterm*, any programs running in the shell will also continue.

If you've used other window managers, you may notice that icon symbols generated by *mwm* are somewhat larger and more decorated. The detail on icons is another aesthetic advantage of *mwm*.

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

A complete list of standard X cursors is shown in Appendix D. OSF/Motif provides some additional cursors. Some of the most common standard cursor shapes, as well as two Motif-specific cursors, are shown in Figure 1-3. As we'll see later, some applications allow you to select the cursor to use.

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


X cursor which appears in the root window.

I-beam cursor, which appears within xterm windows.

Stop cursor displayed when you try to restart mwm from the Root Menu.

Upper-left and lower-right corner cursors, used to resize windows under mwm.

Cross cursor, which is used to move windows under mwm.

Hour glass cursor, which seems to be filling up with sand as mwm is started.

Figure 1-3. Some standard cursors and two Motif-specific cursors

You use the pointer to manage windows and icons, to make selections in menus, and to select the window in which you want to input. You can't type in an *xterm* window until you select that window using the pointer. Directing input to a particular window is called *focusing*. When an *xterm* window has the input focus, the window frame and the text cursor are highlighted, as in Figure 1-4. (The highlighted cursor is a characteristic of *xterm*; other applications may not have a cursor, or may not highlight it to indicate focus.) The window to which input is directed is often called the *active* window.

Be aware that the frame may be highlighted in different ways, depending on the version of *mwm* you are running and the color resources specified for your system. The frame may change from black to white, from grey to white, etc. In any case, the active window's frame will be a different color than the frames of all other windows on the display.

Most window managers require you to select the active window in one of two ways: either by moving the pointer so that it rests within the desired window or by clicking the pointer on the window. By default, the Motif window manager requires you to click the pointer on the window to which you want to direct input. This focusing style is commonly referred to as "click-to-type" or ("explicit focus"). However, as we'll see in Chapter 13, *mwm* can be customized to allow you to direct input focus simply by moving the pointer. This focusing style is commonly referred to as "real-estate-driven" (or "pointer focus").

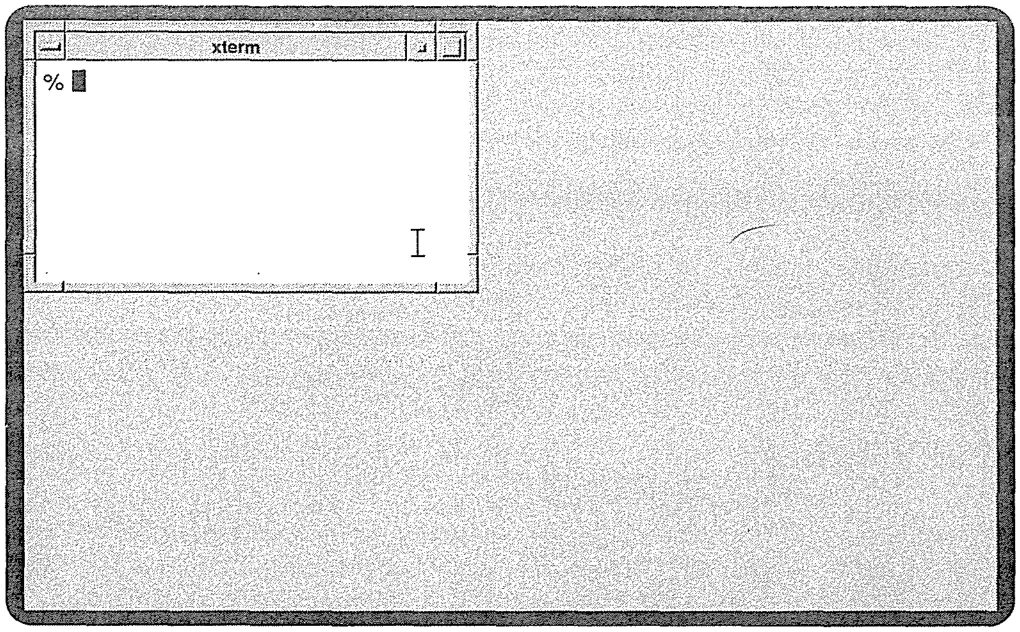


Figure 1-4. Focus on an xterm window

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

As previously mentioned, by default the Motif window manager uses the click-to-type focusing style: you must click the pointer on a window to focus input on that window. When you begin using X with *mwm*, you'll need to select the window to receive input by placing the pointer anywhere within the window and clicking the first (generally the leftmost) button. Once you select the focus window in this way, all input is directed to that window until you move the pointer and deliberately click on another window.

*In a few cases, the window manager may interpret these keystrokes. For example, you can customize *mwm* to display a menu when you type certain keystrokes while the pointer rests on the root window. See Chapter 13, *Customizing mwm*, for more information about mapping window manager functions to certain keys and pointer actions.

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

We find click-to-type focus somewhat laborious. However, a touch typist, who is not inclined to look at the screen, might consider pointer focus too risky. It's possible to knock the pointer out of a window and lose a large amount of text before noticing. Another disadvantage of pointer focus is that it sometimes takes a moment for the input focus to catch up with the pointer, especially on slower machines. If you type right away, some keystrokes may end up in the window you left rather than in the new window. This is actually a bug that happens because of the additional overhead involved in complex window managers such as *mwm* or *twm*. Since you can change the focus policy rather simply, you might want to experiment with both methods. For now, we'll assume you're using the default click-to-type focus.

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

The pointer is also often used to display menus. Some X programs, notably *mwm* and *xterm*, have menus that are displayed (or *posted*) by keystrokes and/or pointer button motions. More versatile than many other window managers, *mwm* provides two default menus, each representing a different menu "style."

The Window Menu is a "pull-down" menu that can be displayed on any window by placing the pointer on the square button in the upper-left corner of the frame (decorated by a short horizontal bar in its center) and either clicking or pressing and holding down the first pointer button. Roughly defined, a pull-down menu is accessed from a graphical element that is always displayed, such as a command button or a menu bar. Figure 1-5 shows an *xterm* window with the Window Menu displayed by clicking the first pointer button in the menu button on the frame.

As you might infer from some of the menu items, you can use the Window Menu to move, resize, and otherwise manage the window on which it is displayed. When you display the Window Menu by clicking the first pointer button (as opposed to pressing and holding it down), the first item available for selection is surrounded by a box. In this case, the first available selection is Move. The first item on the menu, Restore, is used to change an icon back into a window; therefore, it is not useful at this time. The fact that Restore is not selectable is indicated by the fact that it appears in a lighter typeface. The Window Menu is discussed in detail in Chapter 4, *More about the mwm Window Manager*.

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

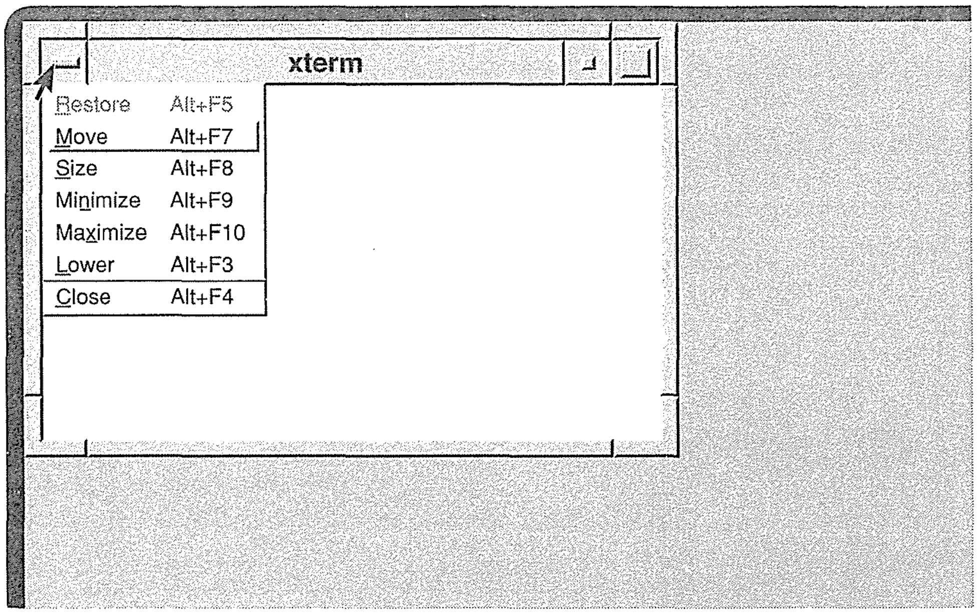


Figure 1-5. A pull-down menu: mwm's Window Menu

In Figure 1-6, the arrow next to the menu title represents the pointer. As you drag the pointer down the menu, each of the menu selections is highlighted. Regardless of the program, you generally select a menu item by dragging the pointer down the menu, highlighting the item you want, and releasing (or sometimes clicking) the pointer button. (*mwm* generally highlights an item by placing a rectangular box around it.) The Root Menu provides commands that can be thought of as affecting the entire display (as opposed to a single window). For example, the first menu item, New Window, creates a new *xterm* window on the local machine and display.

Though *mwm*'s menus can be useful, you'll probably find that you perform most window management functions simply by using the pointer on the window frame. In Chapter 3, *Working in the X Environment*, we'll describe several of these functions. Keep in mind, however, that both of the menus can be useful in certain circumstances. For instance, the Window Menu may be useful when parts of the window frame are obscured by another window. The Root Menu can be customized to execute system commands, such as the *xterm*

*With Motif applications, you generally use pointer button 3 to display pop-up menus. Standard X clients many require another button (often the first). (See "Standard X Clients Versus Motif Clients" later in this chapter.) Chapter 9, *Working with Motif Applications* explains Motif pointer button conventions. Pointer commands for the standard X clients appear in the relevant tutorials throughout this guide.

command initialized by the New Window menu item. It's fairly simple to add items to the Root Menu; you might want to add menu items to start some of the applications you use regularly. Chapter 13, *Customizing mwm*, describes how to add menu items and to perform a variety of modifications.

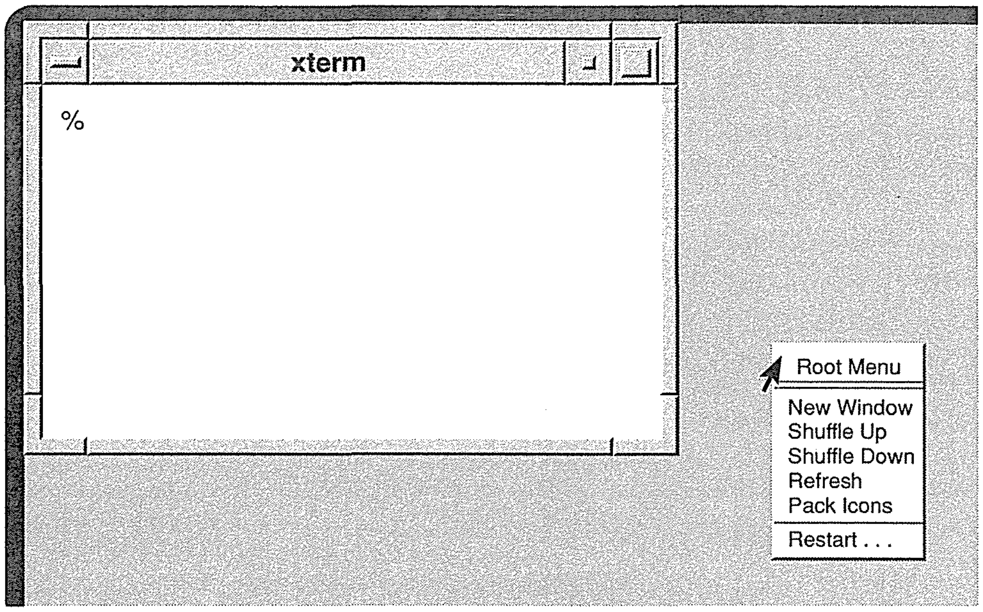


Figure 1-6. A pop-up menu: mwm's Root Menu

As we'll see in Chapter 8, *Other Clients*, some programs provide menus that you can display simply by placing the pointer on a particular part of the window, e.g., a horizontal menu bar across the top.

Some Motif applications offer a variation of a pull-down or pop-up menu that you can display and *leave displayed* while you continue to work. Such a menu is called a "tear-off" menu because it appears to be perforated at the top. Chapter 9, *Working with Motif Applications*, discusses how to tear off one of these menus, select options, and also remove the menu.

A final note about the X display: in X, the terms *display* and *screen* are not equivalent. A display may consist of more than one screen. (However, each display has only one keyboard and pointer.) A multiple screen display might be implemented in several ways. There might be two physical monitors, linked to form a single display, as shown in Figure 1-7. Alternatively, two screens might be defined as different ways of using the same physical monitor. For example, on the Sun-3/60 color workstation, screen 0 is color, and screen 1 is black and white. Each screen is the size of the monitor; you can only view one screen at a time. In practice, the two screens seem to be side by side: you can "scroll" between them by moving the pointer off either horizontal edge of the screen. By default, windows are always placed on screen 0 but you can place a client window on screen 1 by specifying the screen number in the `-display` option when starting the client. (See Chapter 3 for instructions on using the `-display` option.)

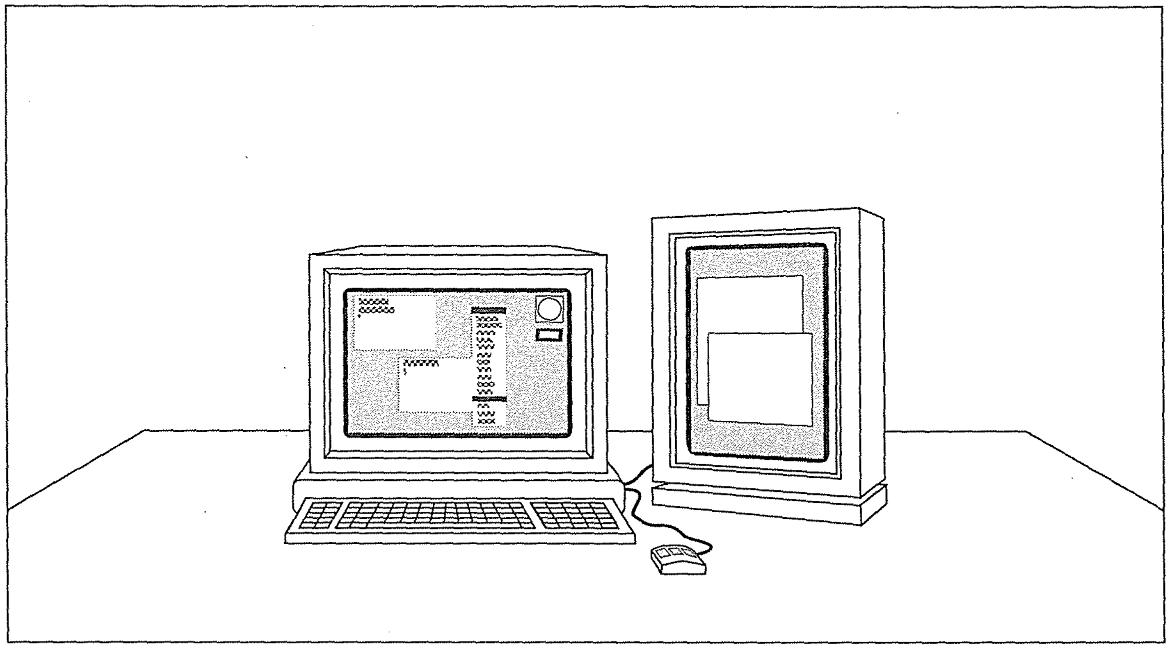


Figure 1-7. A display made up of two physical screens

Standard X Clients versus Motif Clients

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

However, the look and feel of an application is not wholely determined by the window manager. In addition, the programming routines used to create the application also distinguish it. With the exception of *mwm*, all of the applications we've looked at so far have been written (or rewritten) using what is known as the X Toolkit, developed at MIT.

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

Remember that X does not provide a distinct graphical user interface (GUI). X is a basic window system upon which almost any style of GUI can be built. The X Toolkit provides a simplified approach to creating graphical user interface components—guidelines for writing and implementing widgets—rather than offering a set of components with a predefined look and feel. (However, the Athena widget set does provide X Toolkit applications with certain common features, many of which are mentioned in Chapter 8.)

In response to the need for a graphical user interface for X, the Open Software Foundation developed the Motif Toolkit. The Motif Toolkit is based upon the Xt Intrinsics and upon widget sets originally developed by two OSF sponsor companies, Digital Equipment Corporation and Hewlett-Packard. The Motif widget set was designed to emulate the look and feel of the IBM/Microsoft Presentation Manager, popular in the microcomputer world. An application coded using the Motif Toolkit has a distinct look and feel.

AT&T and Sun Microsystems have also developed a GUI—or more precisely, a specification for a GUI—called OPEN LOOK. At present the two major implementations of the OPEN LOOK specification are Sun's XView toolkit (which is not Xt based) and AT&T's OPEN LOOK widget set (which is Xt based). OPEN LOOK and Motif are the prime contenders to establish a graphical user interface standard in the market.

With a few exceptions (notably *mwm*), most of the clients discussed in this guide are standard X clients shipped by MIT. Most of these clients have been built with the X Toolkit and illustrate the use of many of the Athena widgets. When you run these standard clients with the *mwm* window manager, your environment is something of a hybrid—neither a vanilla X nor a pure Motif environment (with Motif applications in addition to the window manager).

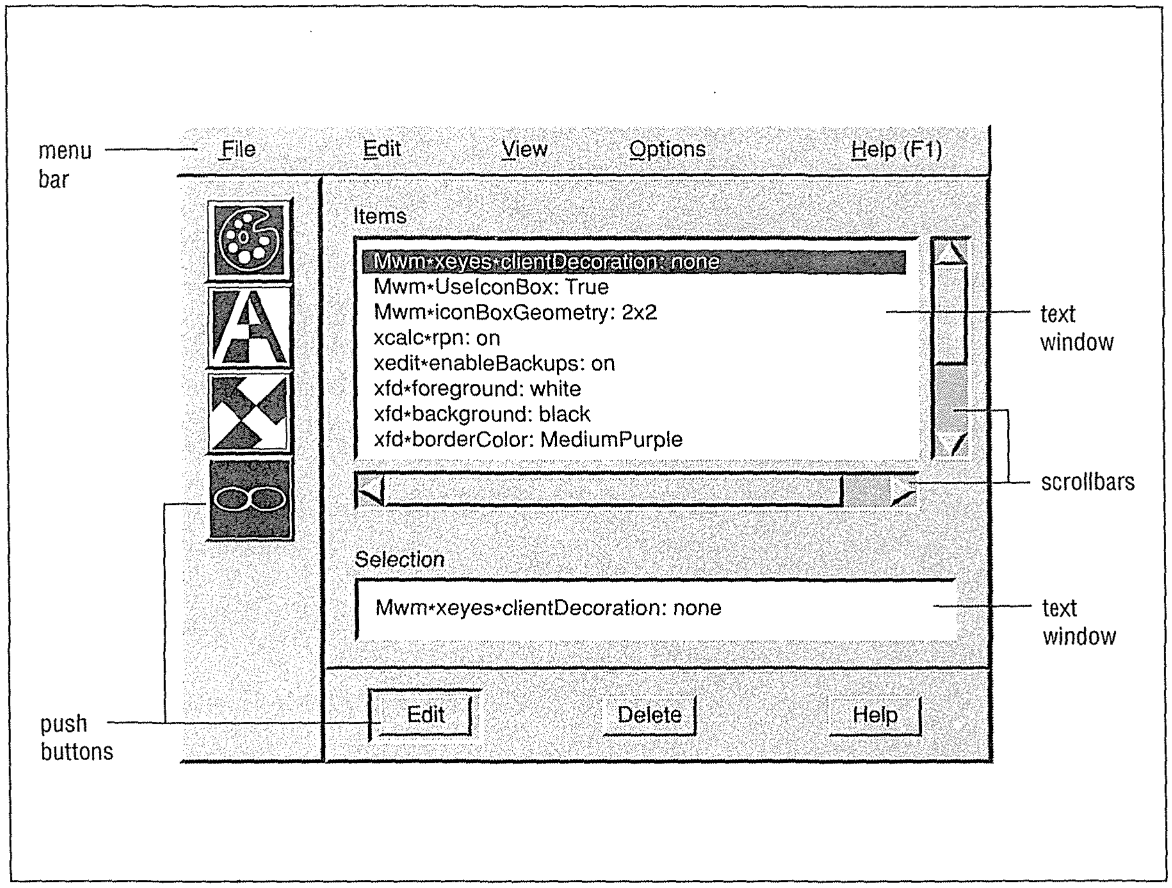
A standard X client running with *mwm* is different from a true Motif application, one coded using the Motif Toolkit. At first look, they may seem very similar—when *mwm* is running all clients on the display are framed in the same way. In addition, certain graphical features provided by the Motif widgets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

Without dissecting every component or closely examining how it functions, let's briefly compare a standard X application to a Motif application, highlighting some of the major differences (primarily in appearance). Many features of Motif and standard X applications also *operate* differently. We'll examine the functionality of various Motif and Athena widgets in more detail in Chapter 8, *Other Clients*, and Chapter 9, *Working with Motif Applications*.

For the comparison, we'll use a now defunct Motif demo program called *mre*, the Motif resource editor. Developed at OSF by Mitch Trachtenberg, *mre* can be used to edit a resource specification file. The editing help *mre* provides is minimal, but the program clearly demonstrates many of the Motif widgets—as it was intended to do. (*mre* has been eliminated from the Motif distribution in version 1.2, but it is appropriate for our purposes—it illustrates several features of a Motif application within a very compact window.)

The standard X client we're using is *xclipboard*, which you use in concert with *xterm*'s "cut and paste" facility, described in Chapter 5. The *xclipboard* client provides a window in which you can paste multiple text selections and from which you can copy text selections to other windows. Similar to the clipboard feature of the Macintosh operating system, *xclipboard* is basically a storehouse for text you may want to paste into other windows, perhaps multiple times.

The *mre* window in Figure 1-8 contains a resource file to be edited. (Resources are a mechanism that allow you to customize the operation of X clients.) The *xclipboard* window in Figure 1-9 contains a long text selection "cut" from an *xterm* window. Some prominent features of each application are labeled. Both clients are illustrated without the *mwm* frame. (When the window manager is running, the frame creates a superficial resemblance among all clients on the display.)



```
Mwm*xeyes*clientDecoration: none
Mwm*UseIconBox: True
Mwm*iconBoxGeometry: 2x2
xcalc*rpn: on
xedit*enableBackups: on
xfd*foreground: white
xfd*background: black
xfd*borderColor: MediumPurple
```

```
Mwm*xeyes*clientDecoration: none
```

Figure 1-8. A Motif application: mre

The most striking difference between the two clients is simply the amount of detail. Like the Motif window manager, the individual Motif widgets create a three-dimensional appearance. The subwindows labeled Items and Selection seem to be set in to the application window. The push buttons (and drawn buttons) are shaded to suggest that they are raised above the level of the application window. (The drawn buttons also feature bitmapped images, three of them rather elaborate.) The menu bar is shaded to appear raised. The scrollbars have clearly distinguishable components, all of which are shaded and contoured to maintain the 3-D impression.

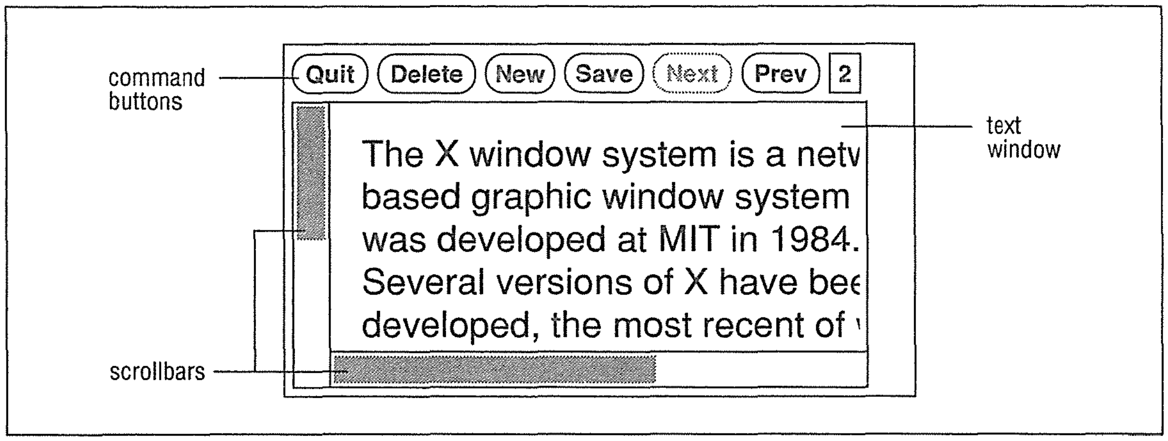

command buttons
Quit
Delete
New
Save
Next
Prev
2
text window
The X window system is a netv
based graphic window system
was developed at MIT in 1984.
Several versions of X have bee
developed, the most recent of
scrollbars

Figure 1-9. A standard X application: xclipboard

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

Another difference is that the *mre* window has a menu bar. Each word on the menu bar is the title of a pull-down menu that you can access by placing the pointer on it and clicking (or pressing and holding down) the first pointer button. The *xclipboard* application doesn't provide any menus—it's a fairly simple program. However, some standard X clients (notably *xman* discussed in Chapter 8) provide pull-down menus accessed by pressing and holding down a pointer button.

Motif pull-down menus generally have a few advantages over pull-down menus provided by standard X clients. While you must press and hold down a pointer button to dislay a menu provided by a standard X client, you can display a Motif menu simply by clicking a pointer button—and the menu stays displayed until you click again. Motif menu items can also be invoked in multiple ways (including pointer actions and keystrokes); the only way to invoke an item from a standard X menu is by dragging the pointer down the menu and releasing the button. The various ways you can work with a Motif pull-down menu are described for *mwm*'s Window Menu in Chapter 4.

Despite differences in general appearance and complexity, *mre* and *xclipboard* have many analogous components. Both applications feature a subwindow containing text that can be edited. (*mre* actually has two such windows, labeled Items and Selection.)

Both applications feature buttons: push buttons in the *mre* window; command buttons in the *xclipboard* window. From a user's viewpoint, push buttons and command buttons are functionally equivalent (though you can invoke a push button's function in more ways). The Motif and Athena widget sets simply identify them by different names. The four push buttons on the left side of the *mre* window are actually called drawn buttons. Drawn buttons are just push buttons decorated with a bitmap image rather than a text label.

Both the *mre* and *xclipboard* windows have a horizontal and a vertical scrollbar, which are used to look at text that is currently outside the viewing window. (These scrollbars are only displayed when the text read into the window extends beyond the bounds of the viewing area. If the text only exceeds the viewing area in one direction—either horizontally or vertically—only one scrollbar will be displayed.) The Athena scrollbar is basically rectangular (actually one rectangle within another). The Motif scrollbar is somewhat more elaborate. Notice the arrows on either end, for instance. These arrows are the hallmark of a Motif scrollbar and can help you readily identify a Motif application. The arrows also provide functionality not duplicated by the Athena scrollbar.

Most of this guide will help you master the basics of working with the MIT client programs running under the *mwm* window manager. Chapter 9, *Working with Motif Applications*, describes how to use some of the additional features provided by commercial applications built with the Motif widget set.

X Architecture Overview

Most window systems are *kernel-based*: that is, they are closely tied to the operating system itself and can only run on a discrete system, such as a single workstation. The X Window System is not part of any operating system but instead is composed entirely of user-level programs.

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

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

This division within the X architecture allows the clients and the display server either to work together on the same machine or to reside on different machines (possibly of different types, with different operating systems, etc.) that are connected by a network. For example, you might use a relatively low-powered PC or workstation as a display server to interact with clients that are running on a more powerful remote system. Even though the client program is actually running on the more powerful system, all user input and displayed output occur on the PC or workstation server and are communicated across the network using the X protocol. Figure 1-10 shows a diagram of such a network.

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

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

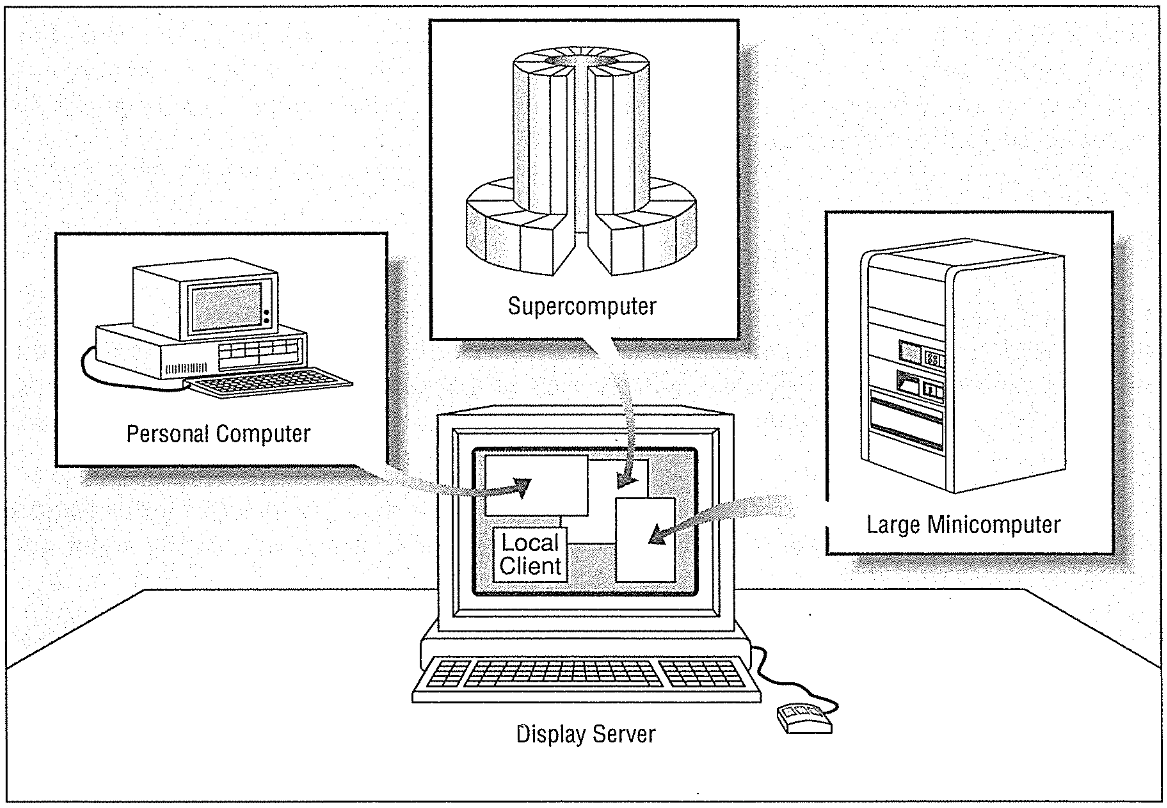


Figure 1-10. A sample X Window System configuration

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

The X Display Server

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

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

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

Servers are available for many types of systems, including PCs, workstations, and X terminals, which may have the server downloaded from another machine or stored in ROM.

Clients

As previously mentioned, a client is an application program. The standard release of X from MIT includes more than 50 client programs that perform a wide variety of tasks. X allows you to run many clients simultaneously: each client displays in a separate window. For example, you could be editing a text file in one window, compiling a program source file in a second window, reading your mail in a third, all the while displaying the system load average in a fourth window.

While X clients generally display their results and take input from a single display server, they may each be running on a different computer on the network. It is important to note that the same programs may not look and act the same on different servers since X has no standard user interface, since users can customize X clients differently on each server, and since the display hardware on each server may be different.

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

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

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

See the *xprop* reference page in Part Three of this guide, and Volume One, *Xlib Programming Manual*, for more information about properties and the *xprop* client.

Several of the more frequently used client programs are discussed in the following sections.

The Window Manager

The way a kernel-based window system operates is inherent in the window system itself. By contrast, the X Window System concentrates control in a special kind of client application called a window manager. The window manager you use largely determines the look and feel of X on a particular system.

The window manager shipped with OSF/Motif is called *mwm*. As we've discussed, *mwm* allows you to move and resize windows, rearrange the order of windows in the window stack, create additional windows, and convert windows into icons, etc. These functions are discussed more fully in Chapter 3 and Chapter 4.

mwm is compliant with the X Consortium's *Inter-Client Communication Conventions Manual* (ICCCM), introduced at Release 3. The ICCCM contains standards for interaction with window managers and other clients. It defines basic policy intentionally omitted from X itself, such as the rules for transferring data between applications, for transferring keyboard focus, for installing colormaps, and so on. If window managers and other applications follow the conventions outlined in the ICCCM, they should be able to coexist and work together on the same server—even if they have been written using different toolkits.

In this guide, we assume you are using *mwm*. Other popular window managers, such as *twm* (the tab window manager), *olwm* (the OPEN LOOK™ window manager), *awm* (Ardent™ window manager), and *rtl* (tiled window manager, developed at Siemens Research and Technology Laboratories, RTL), are also widely used.

If the *mwm* window manager has been customized at your site or you are using a different window manager, many of the concepts should remain the same. However, the actual procedures shown may differ. See Chapter 13, *Customizing mwm*, for a discussion of how to tailor *mwm* to your particular needs.

The xterm Terminal Emulator

X11 itself is designed to support only bitmapped graphics displays. For this reason, one of the most important clients is a terminal emulator. The terminal emulator brings up a window that allows you to log in to a multiuser system and to run applications designed for use on a standard alphanumeric terminal. Anything you can do on a terminal, you can do in this window.

xterm is the most widely available terminal emulator. *xterm* emulates a DEC® VT102 terminal or a Tektronix® 4014 terminal. For each *xterm* process, you can display both types of windows at the same time but only one is active (i.e., responding to input) at a time.

Running multiple *xterm* processes is like working with multiple terminals. Since you can bring up more than one *xterm* window at a time, you can run several programs simultaneously. For example, you can have the system transfer files or process information in one window while you focus your attention on a text-editing session in another window. As you might imagine, having what are in effect multiple terminals can increase your productivity remarkably. See Chapter 3, *Working in the X Environment*, and Chapter 5, *The xterm Terminal Emulator*, for more information about *xterm*.

The Display Manager

The display manager, *xdm*, is a client that is designed to start the X server automatically (from the UNIX */etc/rc* system startup file) and to keep it running. (X can also be started manually, as described in Chapter 2, *Getting Started*.) In its most basic implementation, the display manager emulates the *getty* and *login* programs, which put up the login prompt on a standard terminal, keeping the server running, prompting for a user's name and password, and managing a standard login session.

However, *xdm* has far more powerful and versatile capabilities. From a user's perspective, you can design your own session, automatically running several clients and setting personal resources (such as keyboard, pointer, and display characteristics). Resources are introduced in Chapter 3, *Working in the X Environment*, and discussed fully in Chapter 11, *Setting Resources*.

The system administrator can customize special *xdm* files to manage several connected displays (both local and remote) and to set system-wide X resources (for example, client default features). See Volume Eight, *X Window System Administrator's Guide*, for a discussion of how to set up and customize the *xdm* display manager. Keep in mind that many commercial vendors provide alternative display/session managers. If you are using a display manager other than *xdm*, many of the concepts should remain the same. However, the actual setup procedures may differ.

Other Standard X Clients

The standard distribution of X from MIT includes more than 50 client applications. The client you will probably use most frequently is *xterm*. We've grouped some of the other more useful applications as follows:

Desk accessories

xbiff	Mail notification program
xclock, *oclock*	Clock applications
xcalc	Desktop calculator
xload	System load monitor
xman	Manual page browser

Display and keyboard preferences

xset	Allows you to set various display and keyboard preferences, such as bell volume, cursor acceleration, and screen saver operation.
xsetroot	Allows you to set the appearance of the root window. You might specify a particular color, bitmap graphic image, etc.
xmodmap	Allows you to map keyboard keys and pointer buttons to particular functions.

Font utilities

fs	The font server, a Release 5 innovation, allow you to access fonts over the network. (Fonts no longer need to be resident on your local machine.)
xlsfonts	Lists fonts available on the local machine.
fslsfonts	Lists fonts available via the font server.
xfd	Displays the characters in a single font.
xfontsel	Allows you to display multiple fonts sequentially and select a font to be used by another application.
showfont	Displays the contents of a compiled font file in an (readable) ASCII form. Can be used to convert a font character to a bitmap image using *atobm*.

Graphics utilities

bitmap	Bitmap editor.
xmag	Magnification utility.
atobm, *bmtoa*	Programs to convert ASCII characters to bitmaps and bitmaps to ASCII characters.

Printing applications

xwd	Dumps the image of a window to a file.
xpr	Translates an image file produced by *xwd* to PostScript® or another format, suitable for printing on a variety of printers.
xwud	Redisplays a window dump file created using *xwd*.

Removing a window

xkill	Terminates a client application.

Resource management

xrdb	The X resource database manager allows you to load customized client preferences into the server.
appres	Lists the resources that might be applied to a particular client.
editres	Tests and edits resource specifications.

Window and display information utilities

xlsclients	Lists the clients running on the display.
xdpyinfo	Lists general characteristics of the display.
xwininfo	Lists general characteristics of a selected window.
xprop	Lists the properties associated with a window.

These and other client applications are described in Chapters 5 through 8, and in Chapter 14. In addition, a reference page describing each client and listing its options appears in Part Three of this guide. As more commercial and user-contributed software is developed, many more specialized programs will become available.

Customizing Clients

Most X clients are designed to be customized by the user. A multitude of command-line options can be used to affect the appearance and operation of a single client process. A few of the more useful command-line options are introduced in Chapter 3. Chapter 10 discusses some additional standard options. Part Three of this guide includes a reference page for each client that details all valid options.

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

Generally, these resource values are loaded into the server using a program called *xrdb* (*X* *r*esource *d*ata*b*ase manager). Then the values are accessed automatically when you run a client. Storing your preferences in the server with *xrdb* also allows you to run clients on multiple machines without maintaining an *.Xresources* file on each machine.

There is a separate customization file for the *mwm* window manager called *.mwmrc*, which is also kept in your home directory. By editing the *.mwmrc* file, you can modify several aspects of the window manager's operation, such as the contents of menus, and the key and pointer button sequences used to invoke actions. See Chapter 13, *Customizing mwm*, for more information.

Client customization is introduced in Chapter 3, *Working in the X Environment*, and described fully in Part Two of this guide.

2

Getting Started

This chapter helps you start the X server, the first xterm *(terminal emulator) window, and the* mwm *window manager. These processes may be started automatically when you log in, or you may have to start them manually.*

In This Chapter:

2
Getting Started

Before you can begin using the X Window System, you must do three things:

- Start the X server.
- Start at least one *xterm* terminal emulator.
- Start a window manager. (Although you *can* run X without a window manager, this is fairly limiting.)

Depending on how X is configured on your system, some or all of these steps may be performed for you automatically. This chapter explains how you can tell if the X server, an *xterm* window, and the *mwm* window manager are being started automatically. This chapter also describes how to start these processes manually on a stand-alone system such as a UNIX workstation.

After you've started these preliminary processes, skip to the section "Typing In a Window Once mwm is Running," later in this chapter.

Starting X

Depending on how X is being run on your system, the initial screens you see and the way you log in will be slightly different.

On many systems (and on most X terminals) the display manager, *xdm*, starts X and keeps it running. If your system is set up to use *xdm*, you log on in a special window provided for that purpose. If *xdm* is running, you should never have to start X manually.

On other systems (often UNIX workstations), you may be required to log in at a prompt displayed on the full screen. In these situations, X may or may not be started automatically when you log in. If X is not started automatically, you must start it yourself from the command line, as we'll describe later in this chapter.

Be aware that X is very easy to customize. There are countless command options as well as startup files that control the way the screen looks or even what menus a program displays. If X has been customized on your system, or you are trying out X using someone else's system or login account, things may not work exactly as described here. Customizing the X environment is introduced in Chapter 3, *Working in the X Environment*, and described in detail in Chapters 10 through 14.

Logging On in the Special xdm Window

If the display manager, *xdm*, is running X on your system, you'll probably see a window similar to Figure 2-1 when you turn on your terminal.

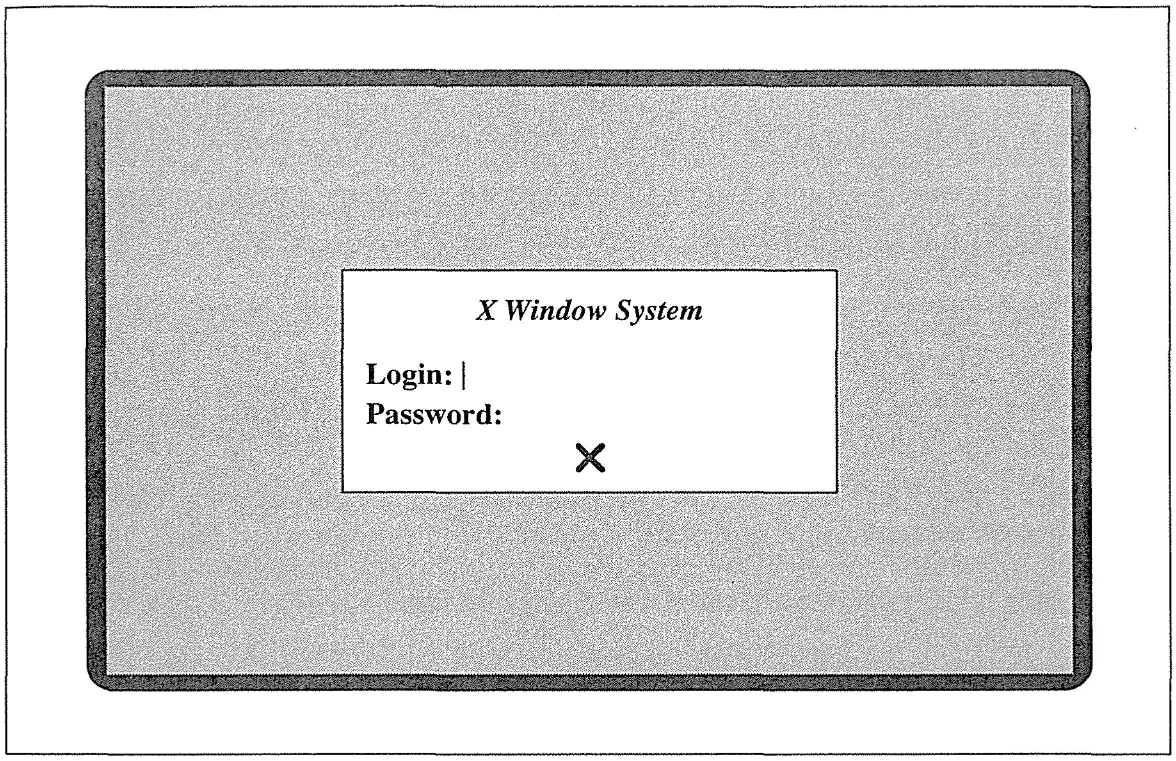


Figure 2-1. xdm login window

Log in just as if you were using a standard alphanumeric terminal. Without any user customization, the display manager executes a standard login "session," providing the first *xterm* window and starting the window manager. The *xterm* window will be displayed in the upper-left corner of the screen. If the Motif window manager is running, you will see the characteristic frame around the *xterm* window, as in Figure 2-2. The name of the application ("xterm") appears in the frame's title area.

The frame provides a quick and easy way to move, iconify, resize, and otherwise manage windows on the screen. Some of the basic window manager functions are introduced in the section "Managing Windows Using the mwm Frame" in Chapter 3.

If the *mwm* window manager is running, skip to the section "Typing In a Window Once mwm is Running," later in this chapter. If the window manager is not running, skip to the section "Bringing Up the Window Manager," for instructions on how to start it.

Be aware that, in addition to *xdm*, other display/session managers are commercially available. For example, Digital Equipment Corporation provides a display manager called *dxsession*, which functions similarly to *xdm*. If your system is using a display manager other than *xdm*, the login procedure may be slightly different. (See Volume Eight, *X Window System*

Administrator's Guide, for instructions on setting up the xdm display manager to run X in a variety of environments.)

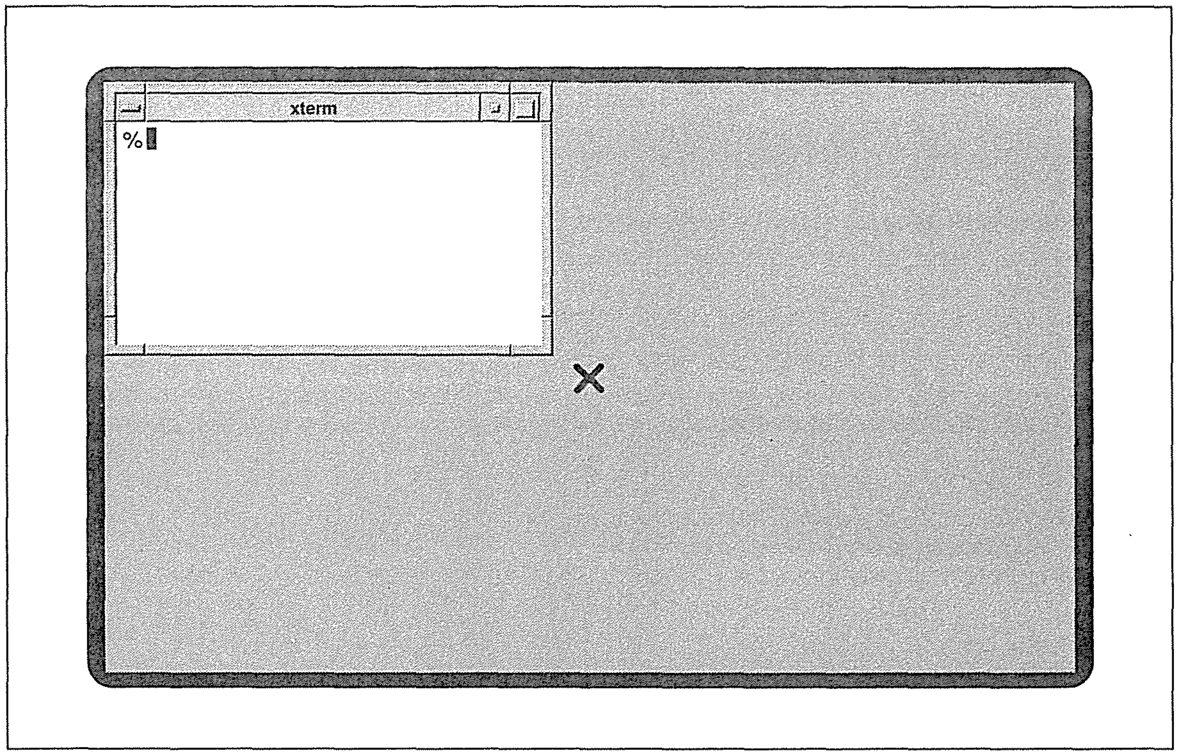


Figure 2-2. Window frame indicates that mwm is running

If you have an *xterm*, but the window manager is not running, skip to the section "Starting the mwm Window Manager" for instructions on how to start it.

As mentioned previously, most X terminals are configured to run X via a display manager, such as *xdm*. When you turn on your X terminal, a login window similar to the one in Figure 2-1 should be displayed to allow you to log in to a local host. If no such login window is provided, there may be a problem with the display manager or the terminal may be configured to connect to the host in another way (commonly via *telnet*). If there's no obvious way to log in on your X terminal, see your system administrator or Volume Eight, *X Window System Administrator's Guide*.

If you are trying to run X on a stand-alone system, such as a workstation, or in a more complex network environment, see Volume Eight for instructions on configuring *xdm*.

Logging In at a Full Screen Prompt

If you log in at a prompt displayed on the full screen, your system is probably set up to work in one of three ways. First, some workstations may be set up to start the server, open up the first *xterm* window, and possibly even start the window manager automatically. If all of these processes are running when you log in, the initial *xterm* window will be framed, as in Figure 2-2. Once the server, initial *xterm*, and the *mwm* window manager are running, you can skip to the section "Typing In a Window Once mwm is Running."

Some systems are set up to start the server and initial *xterm* window when you log in, but not the window manager. If this is the case, your screen should then look something like the one in Figure 2-3.

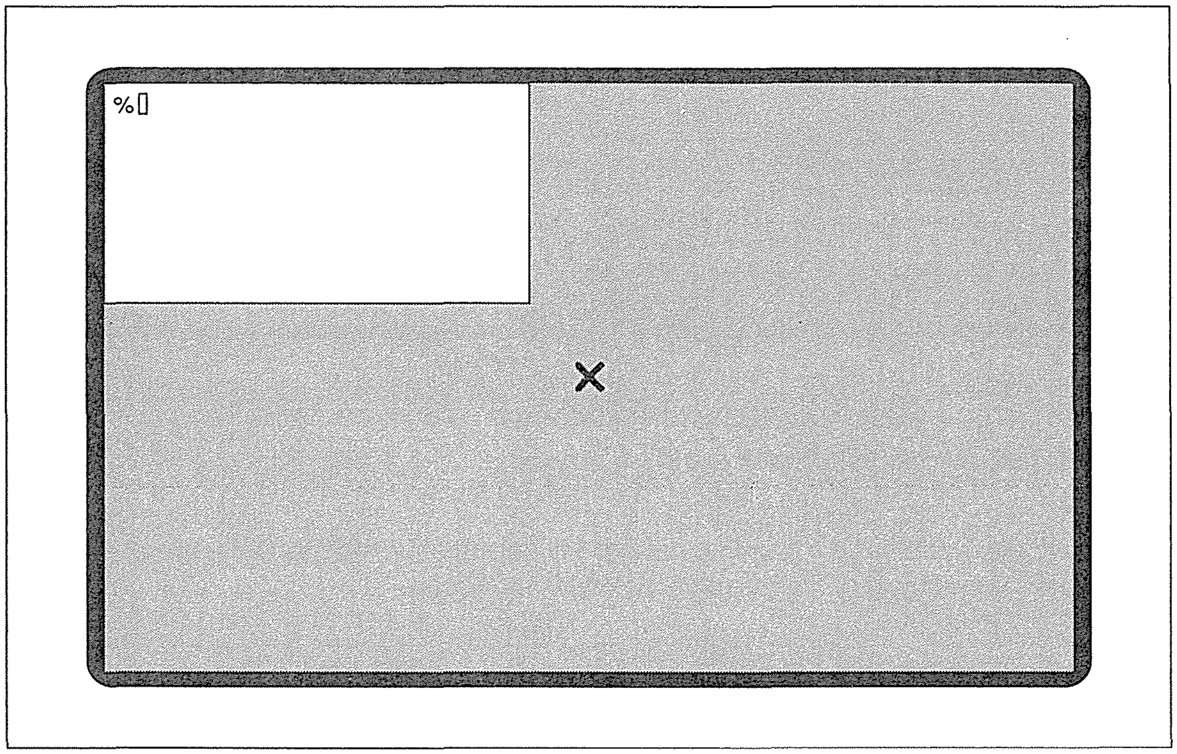


Figure 2-3. Workstation with login xterm window on the root window

If you have an *xterm* window without a frame, you must start *mwm* yourself. Skip to the section "Starting the mwm Window Manager" later in this chapter.

Be aware, however, that on some systems, X is not started automatically. When you log in at the prompt displayed on the full screen, no windows are opened; instead the entire screen functions as a single terminal, as shown in Figure 2-4.

```
%
```

Figure 2-4. Workstation functioning as a single terminal: X isn't running

If no windows are displayed when you log on at a full screen prompt, you must start the X Window System manually.

Starting X Manually

To start X manually, first make sure that the X11 directory containing executable programs is in your search path.* If not, add the pathname */usr/bin/X11* to the path set in your *.profile* or *.login* file.

If another windowing system (such as SunView™) is running, you must first kill it.

Then at the system prompt, enter:

```
% xinit
```

*For more information on how to set your search path, see Appendix A, *Managing Your Environment*. Note that the appropriate pathname to add may be different in vendor distributions.

The *xinit* program starts the X server and creates the first *xterm* window in the upper-left corner of your display.* The *xterm* window indicates that X is running on your display. (In Appendix A, *Managing Your Environment*, we'll show you a simple way to run *xinit* automatically when you log in. For more information on automating X, see Volume Eight, *X Window System Administrator's Guide*.)

Starting the mwm Window Manager

To bring up the *mwm* window manager, you must enter the *mwm* command in the login *xterm* window. As described in Chapter 1, *An Introduction to the X Window System*, you can only provide input to one window at a time. X does not automatically know which window you want to type in, even if only one window appears on the display, as in Figure 2-3. In order to type in the *xterm* window on the display, you must first focus input to that window. The window to which input is focused is often referred to as the active window.

When no window manager is running, you focus input simply by moving the pointer so that the cursor symbol on the screen rests within the window. Depending on where the pointer rests, the cursor symbol representing it on the screen differs. When the pointer rests on the root window, it appears as a large X (another indication that X is running). When you move the pointer into the *xterm* window, it should change to what is known as the I-beam cursor symbol, which looks like a very thin letter "I".

When you move the pointer into the *xterm* window, input is focused there. Notice that the rectangular text cursor changes from an outline to a solid color. The change in the text cursor is another indication that input is focused on the window. Once you've selected the *xterm* window as the active window, the characters you type will appear on the window's command line.

Start the *mwm* window manager by typing:

```
    % mwm &
```

The screen will momentarily go blank. Then while *mwm* is starting up, the root window pointer changes to an hour glass that appears to be filling up with sand. When the hour glass is full, the *xterm* window will be redisplayed, this time with the characteristic frame, indicating that *mwm* is running.

*If *xinit* produces a blank background with no terminal window, software installation was not completed correctly. Reboot your workstation and try again. Before invoking *xinit*, look in the directory */usr/bin/X11* for a file whose name begins with a capital X but otherwise has a similar name to your workstation (e.g., Xsun). When you find one that seems a likely possibility, try this command:

```
    % xinit -- Xname
```

If that works, link X*name* to X, and *xinit* will work correctly thereafter. For example:

```
    % cd /usr/bin/X11
    % ln Xsun X
```

Note that it is important to run *mwm* in the background by typing an ampersand (&) at the end of the command line so you can continue to enter additional commands in the *xterm* window. If you neglected to do this on a system that supports job control, type Control-Z to suspend *mwm*, then use the *bg* command (see *csh*(1)) to place it in the background.

If the system you're on does not support job control, interrupt the process using the appropriate key sequence (Control-C on many systems) and start over.

Typing In a Window Once mwm is Running

Now that you've started the X server, the first *xterm* window, and the *mwm* window manager, you'll want to proceed by entering commands in the *xterm* window. However, if you type some characters, you'll find that the keystrokes do not appear in the *xterm*—or anywhere else on the display!

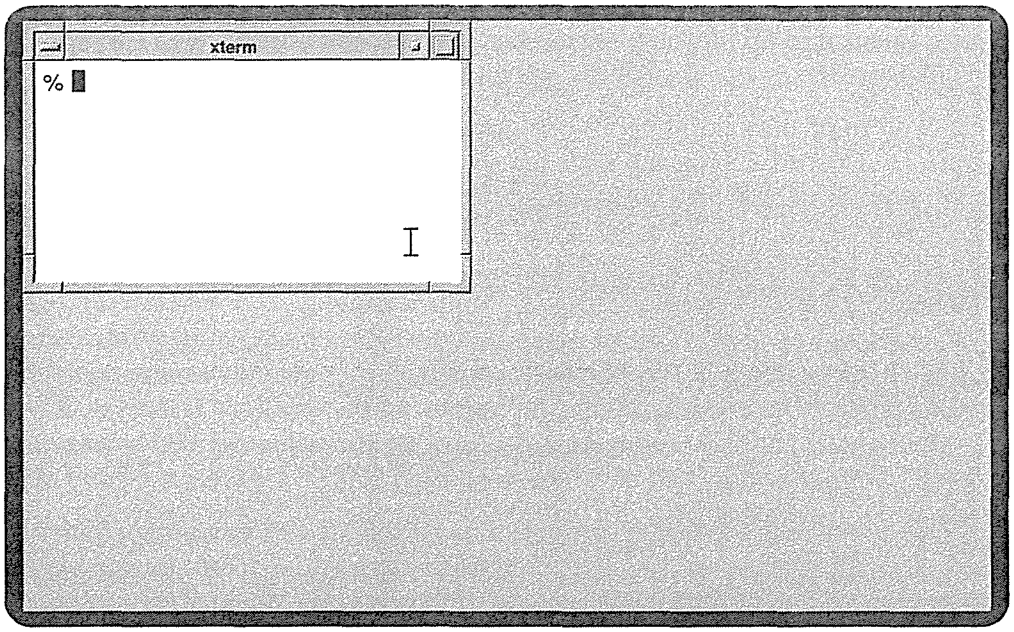
In order to type in a window when *mwm* is running, you must first move the pointer into the window and click the first button. As described in Chapter 1, this focus policy is commonly referred to as click-to-type or explicit focus. (When *mwm* is not running, you select the window to receive input—the active or focus window—simply by moving the pointer into that window.) Until you select the *xterm* window as the window to receive input, any keystrokes are lost.

This feature of *mwm* can be even more confusing if *mwm* is not started automatically when you log in. Before you start the window manager, you can focus input on a window simply by moving the pointer. As described in the preceding section, you must move the pointer into the first *xterm* window in order to enter the *mwm* command. However, once you run *mwm*, the focus policy changes to click-to-type. If you try to type additional characters without clicking on the window, the keystrokes will be lost.

To select the focus window when *mwm* is running:

1. Move the pointer into the *xterm* or other client window.
2. Click the first pointer button. (To click a pointer button, you press it down and release it without pausing. Pointer actions are explained in the section "Using the Pointer" in Chapter 3.)

You can tell which window has the input focus by several changes in the appearance of the display, illustrated in Figure 2-5 and described below. (Once you are running several windows simultaneously, it's important to be able to identify the focus window easily.)



```
% █
```

Figure 2-5. Changes in appearance indicate a window has the input focus

First, the color of the window frame will change in some way. As described in Chapter 1, *An Introduction to the X Window System*, the active window's frame may be highlighted in different ways, depending on the version of *mwm* you are running and the color resources specified for your system. The frame may change from black to white, from gray to white, etc. In any case, the active window's frame will be a different color than the frames of all other windows on the display.

The cursor symbol that represents the pointer will also change. Depending on where the pointer rests, the cursor symbol representing it on the screen differs. When the pointer rests on the root window, it generally appears as a large X (another indication that X is running). When you move the pointer into the *xterm* window, it should change to what is known as the I-beam cursor symbol, which looks like a very thin letter "I".

In the *xterm* application window itself, the rectangular text cursor changes from an outline to a solid color, and the window border is highlighted.

Once you've selected the *xterm* window as the active window, the characters you type will appear on the window's command line.

When you are using explicit (click-to-type) focus and the other default *mwm* resources, selecting a window to receive input also raises that window to the top of the window stack. As we'll see in Chapter 13, *Customizing mwm*, this behavior is controlled by an *mwm* resource variable called `focusAutoRaise`, which is true by default when explicit focus is in effect.

If you are working with a stack of windows that overlap, selecting a focus window automatically raises that window to the top of the stack (in effect the front of the display).

Keep in mind that *mwm* is highly customizable. You can specify dozens of features, including the color of the active window's frame, the options available on menus, and how certain window management functions are invoked. As we've discussed, one of the most significant modifications you can make to *mwm* is to change the focus policy from click-to-type (or explicit focus) to real-estate-driven (or pointer focus).

3

Working in the X Environment

This chapter shows you how to begin working productively in the X environment. It describes how to:

- *Open a second* xterm *window.*
- *Move windows; raise windows to the front of the display; convert windows to icons.*
- *Close an* xterm *window.*
- *Start other clients in convenient places on the display.*
- *Run clients on remote machines.*
- *Customize a single client process using command-line options.*
- *Specify alternate default characteristics for a client using resource variables.*

In This Chapter:

☞

3
Working in the X Environment

At the end of the last chapter, you should've had the X server, the first *xterm* window, and the *mwm* window manager running. The current chapter illustrates some of the system's basic capabilities so you can begin working more productively. This chapter shows you how to:

- Open a second *xterm* window.
- Use the pointer to affect windows on the display.
- Iconify, deiconify, maximize, raise, move, resize, and close windows using the pointer on the *mwm* frame.
- Close an *xterm* window from the command line.
- Start additional client programs, on both local and remote machines.
- Organize the display.

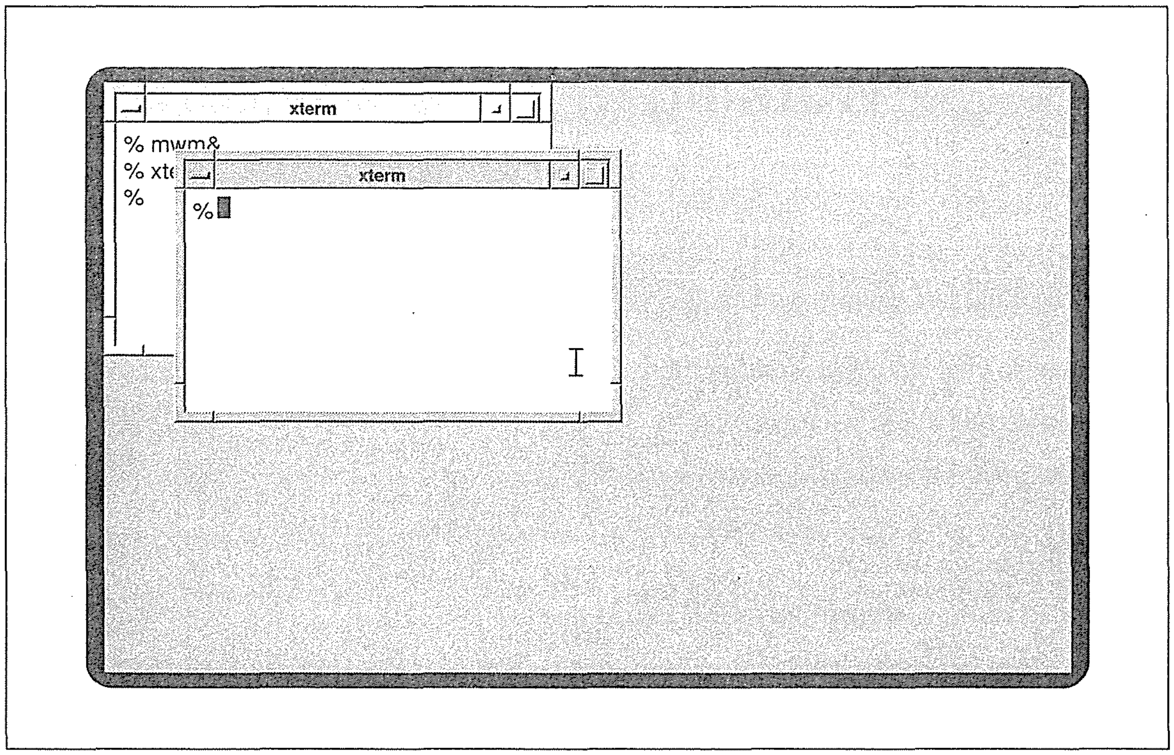
This chapter also introduces some basic ways to customize X clients to better suit your needs.

Creating Other Windows

Once you focus input on the first *xterm* window, as described in Chapter 2, you can enter commands to open other client windows. For example, you can open a second *xterm* window by typing this command at the prompt in the first *xterm* window:

```
    % xterm &
```

After a few moments, a second *xterm* window will be displayed on the screen. As we'll see later in this chapter, you can specify the location for a new window using a command-line option (or in many cases using a resource variable stored in a file in your home directory). If you don't specify position on the command line (or in a resource file), by default *mwm* automatically places new windows offset from the upper-left corner of the screen, as shown in Figure 3-1.*



```
% mwm&
% xte
%
```

```
%
```

Figure 3-1. mwm automatically places the second xterm window

The new *xterm* window displays a prompt from whatever shell you are using. In this case, the new window is running the UNIX C shell.

Notice that the second window's frame is a dark gray, indicating that input is focused on it. The first window's frame has changed from dark to light gray; it no longer has the input focus. It's important to be aware that when you start a new window (and click-to-type focus is being used), the new window automatically takes over the input focus. (Note that the colors may vary according to system defaults.)

In the default *mwm* configuration, to switch back and forth between windows you must move the pointer from one window to the other and click the first button. Notice that if you are working with a stack of windows that overlap, selecting a window as the active window automatically raises that window to the top of the stack (in effect, the front of the display).

*If you start multiple processes in a row, the windows will be placed progressively further from the upper-left corner (towards the opposite corner), so that no window completely overlaps another.

You can customize *mwm* to allow you to place new windows interactively using the pointer. This modification is performed by setting a resource variable called `interactivePlacement` to a value of true. See Chapter 13 for instructions on modifying *mwm*. See the *mwm* reference page in Part Three of this guide for more information about `interactivePlacement`.

Whatever you type will appear in the window with the highlighted frame. Try starting a command in both windows. For example, start up *vi* or another text editor in the second *xterm* window. Notice how you can switch back to the first window to type a new command, by moving the pointer and clicking—even if you leave *vi* in insert mode or some other command in the process of sending output to the screen. Whatever process was running in the window you left will continue to run. If it needs input from you to continue, it will wait.

You must always switch focus to work with multiple windows. However, *mwm* has complicated matters by placing the second *xterm* window in Figure 3-1 in a very inconvenient place. The second window overlies the first window and almost completely obscures it.

Windows commonly overlap on the display. The window manager allows you to change the position and size of windows so that you can work effectively in such situations.

The primary window management tool *mwm* provides is the window frame. The section "Managing Windows Using the mwm Frame," later in this chapter, shows you how to perform these functions simply by using the pointer on various parts of the frame. But before we can learn to perform these window management functions, we need to learn more about pointer actions.

Using the Pointer

As explained in Chapter 1, the cursor on the screen follows the pointer's movement on the desktop. Move your pointer around on the desk to get used to this. Keep in mind that different pointers respond differently. If you move to another display, the screen cursor may move more quickly or slowly than the one you're used to. The *xset* client (described in Chapter 14, *Setup Clients*) lets you modify pointer behavior.

You use the pointer to indicate a graphical element on the screen, such as a window, icon, or command button. Most pointers have three buttons. For our purposes, we'll refer to these buttons as first, second, and third, where the first button is the leftmost on the pointer, the second is in the middle, and third is the rightmost.* By placing the pointer on a particular element and then performing some button action (and possibly a pointer motion), you can invoke a variety of commands. The types of button actions and pointer motions you can perform are:

Click	To click a button, you press the pointer button down and release it. A click is a rapid action; there is no pause between the press and release. A double click is two full clicks in succession, with no pause between clicks. A triple click is three clicks in succession.
Press	To press a button, you push the button down and hold it down.

*Keep in mind that "first" is a logical distinction made by X, not a physical one. The first logical pointer button generally corresponds to the leftmost button on the pointer. (Thus, in some contexts you may find the buttons referred to as left, middle, and right.) However, X allows you to change the correspondence of logical and physical buttons. For example, you can reassign the first logical button to be the rightmost button on the pointer. A lefthanded person might opt to reverse the order of the buttons. You remap pointer buttons using a client called *xmodmap*, which is described in detail in Chapter 14, *Setup Clients*.

Release	After pressing a button down, you release it by letting up on the button.
Drag	To drag a graphical object (such as a window or icon) from one location on the screen to another: place the pointer on the object; press one or more pointer buttons; move the pointer to another location (dragging the object); and release the button(s).

Keep in mind that some commands or actions are invoked by a simple click on a particular graphic element, as illustrated by *mwm*'s click-to-type focus. Alternatively, some actions require a button press and pointer motion (i.e., dragging).

When dragging is used to move an object, the actual object does not appear in the new location until you complete the movement and release the pointer button. Instead, you appear to drag an outline representing the object. When you release the pointer button, the actual object appears in the new location. This effect is illustrated in the section "Moving a Window," later in this chapter.

Dragging is also commonly used to change the size of a window. Again, an outline indicates that the window's size is changing. When the outline approximates the size you want, you release the pointer button and the actual window is redrawn using the selected dimensions.

The following sections describe how to perform the most basic window management functions, which require you to use the pointer in the ways we've discussed.

Managing Windows Using the mwm Frame

Figure 3-2 shows an *xterm* window "framed" by *mwm*. The window frame itself and several features of it are tools that allow you to manage the window using the pointer.

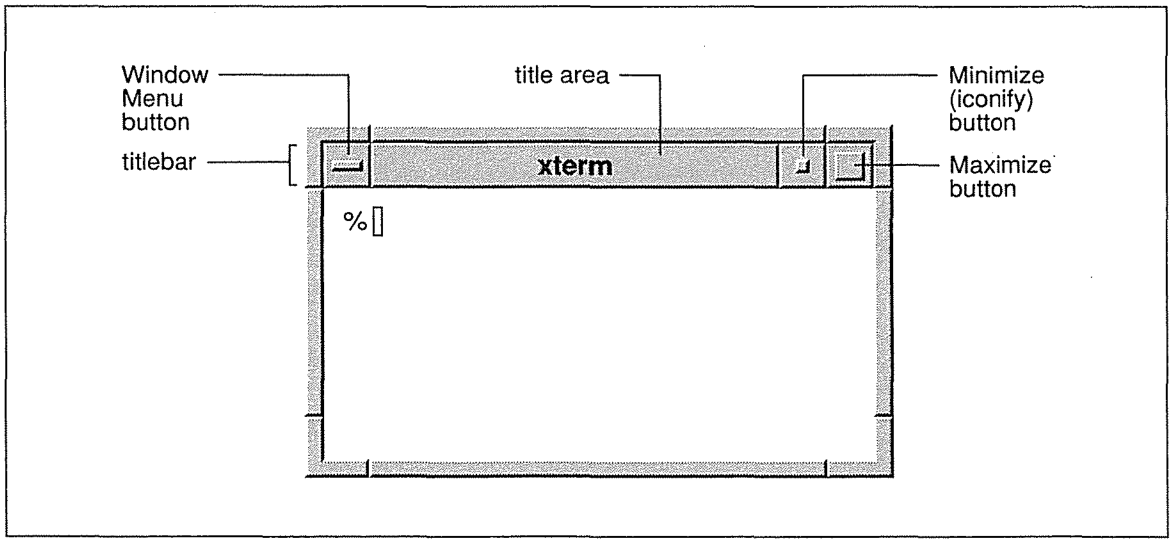


Figure 3-2. An xterm window framed by mwm

The wider top edge of the frame is the titlebar. The titlebar is composed of several parts including a title area (displaying the name of the application) and three command buttons (Minimize, Maximize, and Window Menu). Notice that whenever you move the pointer into the titlebar, the pointer changes to the arrow cursor.

Though it's not apparent from Figure 3-2, you can perform most window management functions by using the pointer on various parts of the frame. The following sections explain how to iconify, maximize, move, raise, resize, and close windows using the frame. Chapter 4 describes menu items and keyboard shortcuts that can be used as alternatives to the frame. These are important when a window's frame is obscured by other features of the display. Chapter 4 also describes additional functions provided by *mwm* menus.

Converting a Window to an Icon

As discussed in Chapter 1, an icon is a symbol that represents a window in an inactive state. There are many circumstances in which it might be desirable to iconify a window:

- To prevent yourself from inadvertently terminating a window, as in the case of the login *xterm*.
- While running a program whose progress you don't need to monitor; if a window is tied up running a process and you don't have to see it, the window is just taking up space.
- If you are only using an application occasionally. For example, you might be running the *xcalc* calculator program, but only using it every so often.
- If you want to use several application windows, but only display a few at a time; this arrangement can be somewhat less confusing than a display crowded with windows. Having some windows iconified also frees you from constantly shuffling the stacking order.

The Minimize command button on the *mwm* frame allows you to convert a window to an icon (iconify it). The Minimize button appears immediately to the right of the title area and is identified by a tiny square in its center. To iconify a window, use the following steps:

1. Place the pointer within the Minimize command button. The pointer simply has to rest within the button's outer border, not within the tiny square identifying it.
2. Click the first pointer button. The window is iconified. Figure 3-3 shows a window being converted to an icon in this way.

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 a window known as an *icon box*. In Chapter 13, *Customizing mwm*, we'll discuss the specifications necessary to set up an icon box.

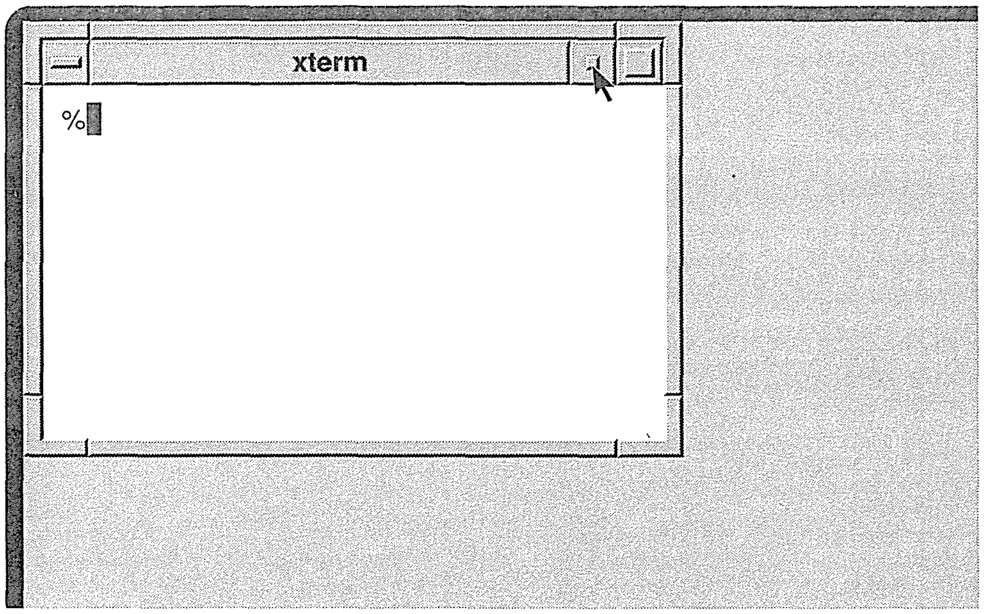


Figure 3-3. Converting a window to an icon with the Minimize button

Converting an Icon to a Window

To convert an icon back to a window (deiconify it), place the pointer on the icon and double click, using the first pointer button. The window is redisplayed in the position it appeared before it was iconfied (and is also raised to the top of the stack).

Between the first and second clicks, you'll probably notice that another small window is displayed for an instant above the icon. This window is actually a menu, called the Window Menu. (This menu can be displayed from a window or from an icon and can be used to invoke several window management functions on the window or icon. We'll discuss the Window Menu in more detail in Chapter 4, *More about the mwm Window Manager*.)

Be aware, however, that if you pause too long between the two clicks in deiconifying a window, the second click will not be interpreted and the icon will not be converted back to a window. Instead the Window Menu will remain on the screen, as in Figure 3-4.

Notice that in addition to displaying the menu, clicking has caused the icon image to change in appearance. The icon label is wider and the label and the frame surrounding the icon are highlighted. These changes indicate that the icon has the input focus; thus the icon will interpret subsequent keystrokes as window manager commands.

Notice also that the first item on the menu, Restore, is surrounded by a box, indicating that it is available for selection. Restore means to restore an icon to a window (or, as we'll see, a maximum size window to its original size). When the Window Menu is displayed above an icon, you can convert the icon to a window by placing the pointer on the Restore menu item and clicking the first pointer button. (Whenever a Window Menu item is boxed, you can also select it by pressing either the Return key or the space bar.)

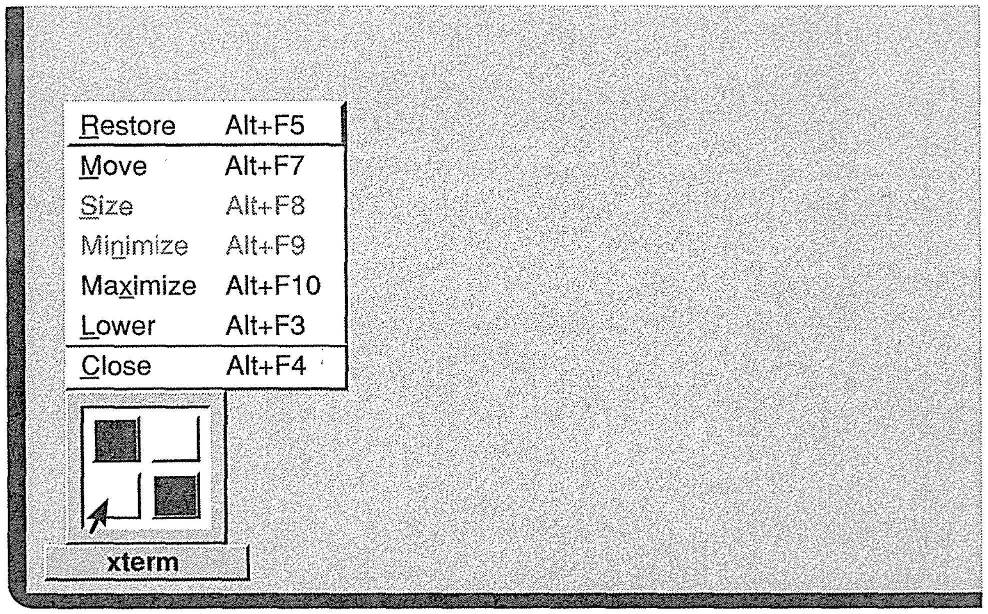


Figure 3-4. Window Menu being displayed over an icon

If you want to remove the menu without invoking a command, simply move the pointer off the icon and menu and click the first pointer button. The menu will be removed; however, the icon will retain the input focus—the label and border will remain highlighted—until you focus input on another window or icon.

Maximizing a Window

Maximizing a window generally means enlarging it to the size of the root window. (In some cases, a client application may specify its own maximum window size and maximizing will produce a window of this size.) This action can be performed using the Maximize command button, which is located to the right of the Minimize command button (in the upper-right corner of the window).

The Maximize command button is identified by a larger square in its center. It allows you both to enlarge the window to the size of the root window (or to the maximum size the client allows), and once it has been enlarged, to convert it back to its original size.

To maximize a window, use the following steps:

1. Place the pointer within the Maximize command button. The pointer simply has to rest within the button's outer border, not within the square identifying it.

2. Click the first pointer button. The window is maximized, as illustrated in Figure 3-5.

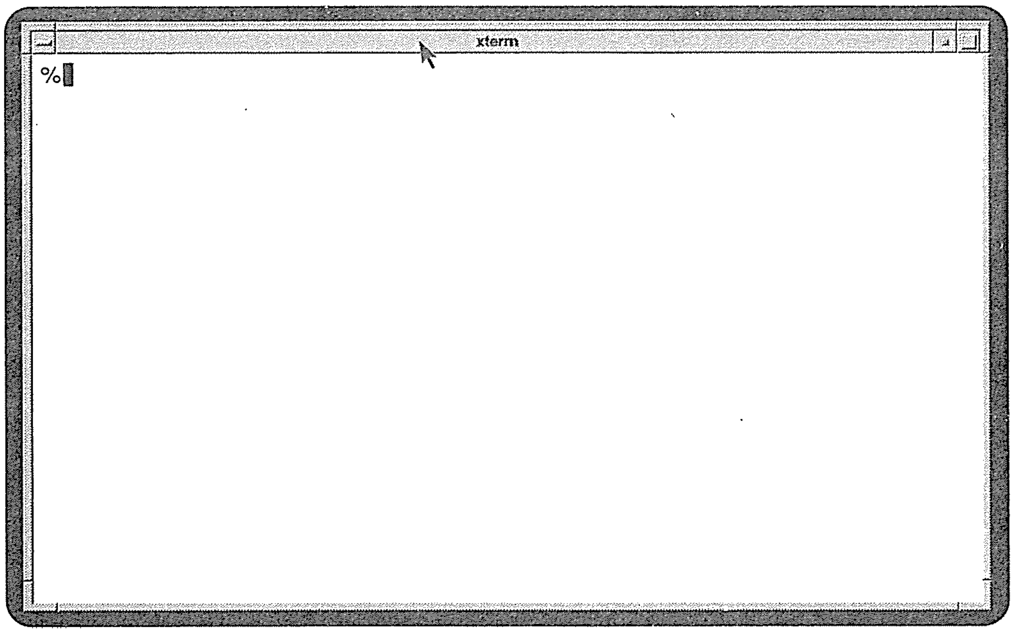


Figure 3-5. A maximized window

The large window should function in the same way it did before it was maximized. Theoretically, you can maximize an *xterm* window to have a single, very large terminal screen. However, be aware that certain programs you may run within an *xterm*, such as the *vi* text editor, do not always work properly within a window of this size (even if you've used the *resize* client, as described in Chapter 5, *The xterm Terminal Emulator*). The Maximize function is more safely used with an application that displays a graphic image or performs a simple function, such as *xclock*.

Also, some client programs that do not support resizing, such as the Release 3 version of *xcalc*, cannot be maximized correctly. In the case of *xcalc*, the frame surrounding the calculator application is maximized, but the actual calculator remains the same size.

The Maximize button is a toggle. To convert a maximized window back to its original size, click on the Maximize button again with the first pointer button. The Restore item on the Window Menu will also perform this function.

Raising a Window or Icon

We've already seen the necessity for raising windows on the display: frequently windows overlap. In order to work with a window that is all or partially covered by another window, you'll want to raise it to the top of the window stack. Using the default *mwm*, you raise a window with the following steps:

1. Place the pointer on any part of the window frame, except the three command buttons (Minimize, Maximize, and Window Menu).
2. Click the first pointer button. The window is raised to the top of the stack.

When you are using explicit (click-to-type) focus, this click also selects the window to receive input, i.e., makes the window the active window. Once you have raised a window to the top of the stack, you should be able to enter input and read output easily.

Windows may obscure icons on the display. (*mwm* does not allow one icon to obscure another.) If an icon is partially visible under a window, you can raise it using the following steps:

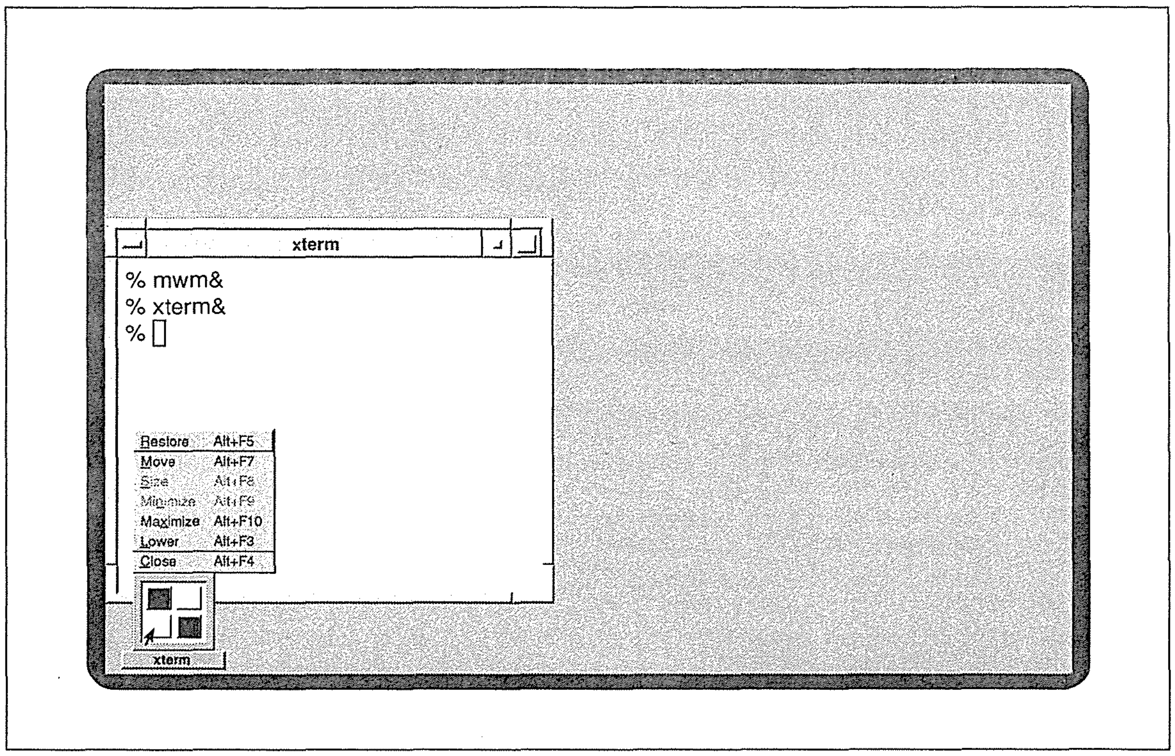
1. Place the pointer on the obscured icon.
2. Click the first pointer button.

The icon is raised to the top of the stack.

Figure 3-6 illustrates an icon being raised in front of a window.

Notice that in addition to being raised to the top of the window stack, a menu (called the Window Menu) is displayed over the icon. (This menu can be displayed from a window or from an icon and can be used to invoke several management functions on the window or icon. We'll discuss the Window Menu in more detail in Chapter 4.) To remove the menu, move the pointer off of the icon and menu and click the first button.

Notice also that the icon image changes in appearance when you raise the icon to the top of the stack (as it did when we paused between the double click to deiconify). The wider label and the highlighted label and frame indicate that the icon has the input focus. (Remember, when an icon has the input focus, it will interpret subsequent keystrokes as window manager commands.) Even when you remove the Window Menu, the icon will retain the focus (and remain highlighted). When you direct focus to another window (or icon), the icon label will become normal again.



```
% mwm&
% xterm&
% ▯
```

Figure 3-6. Raising an icon

Moving a Window

In many cases you'll want to move a window from one location on the display to another. The largest part of the titlebar is known as the *title area*, primarily because it displays the name of the application. The title area allows you to move the window, using the following steps:

1. Place the pointer within the title area. The pointer changes to the arrow cursor.
2. Press and hold down the first pointer button.
3. Move the window by dragging the pointer. Figure 3-7 shows a window being moved in this way. When you begin to move the window, the pointer changes to a cross arrow pointer and a window outline appears. This outline tracks the pointer's movement. In the center of the screen, a small, rectangular box also appears, displaying the x and y coordinates of the window as you move it.
4. Drag the cross arrow pointer with the window outline to the desired location on your screen.
5. Release the first pointer button. The window will move to the selected location.

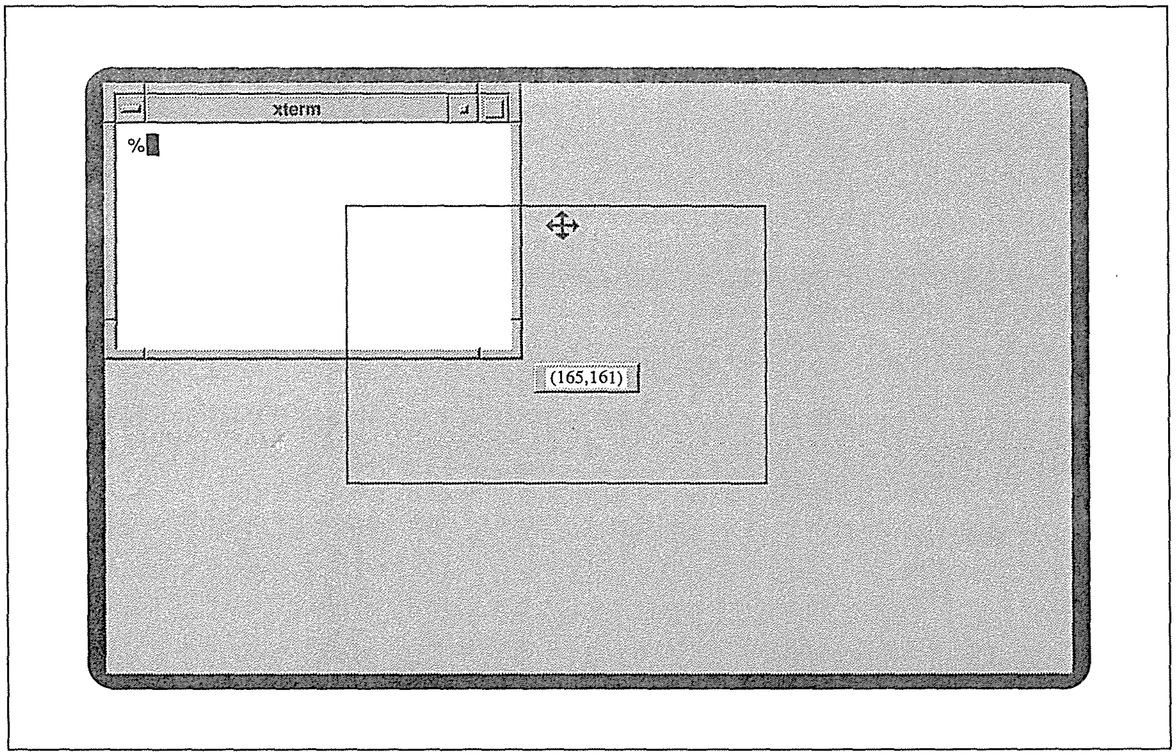


Figure 3-7. Moving a window by dragging the title area

With the default configuration of *mwm*, moving a window also selects that window as the active or focus window.

Moving an Icon

Moving an icon is similar to moving a window. To move an icon:

1. Place the pointer on the icon.
2. Press the first pointer button.
3. Move the icon by dragging the pointer. Figure 3-8 shows an icon being moved in this way. When you begin to move the icon, the pointer changes to a cross arrow pointer and an icon outline appears. This outline tracks the pointer's movement.
4. Drag the cross arrow pointer with the icon outline to the desired location on your screen.
5. Release the first pointer button. The icon will move to the selected location.

With the default configuration of *mwm*, moving an icon also selects that icon to receive the input focus.

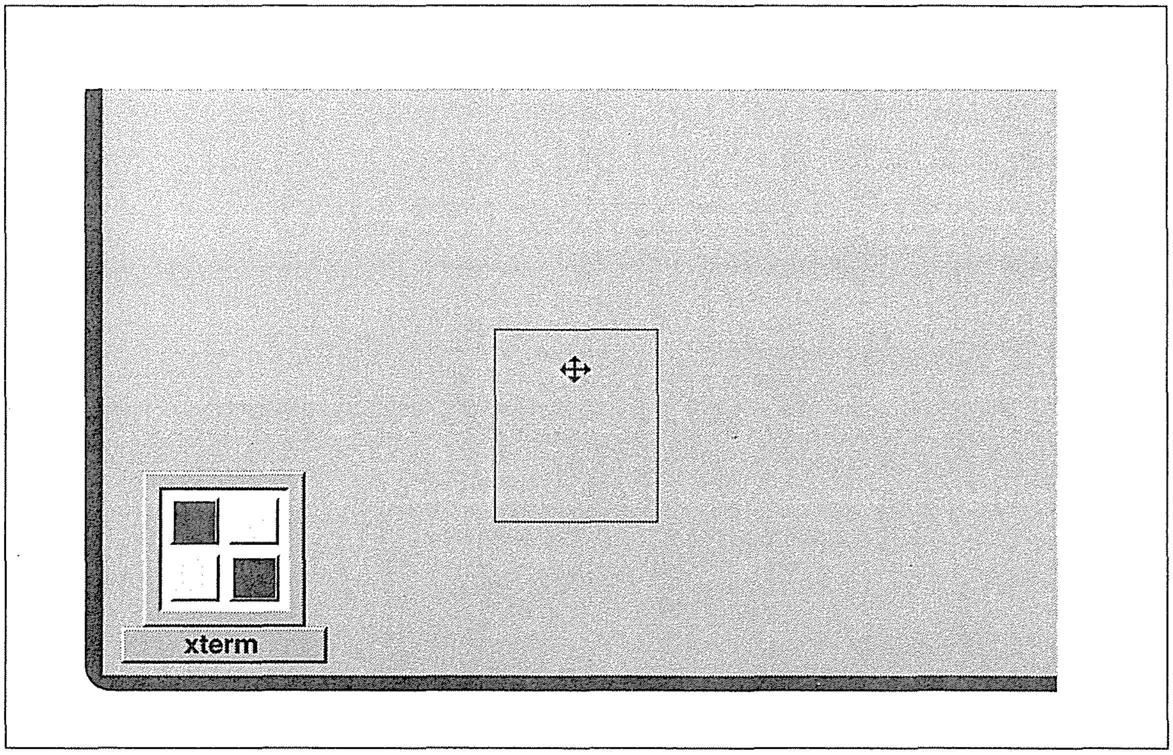


Figure 3-8. Dragging an icon to a new location

Resizing a Window

One of the most distinctive and useful features of the *mwm* window frame is not at all obvious. The entire frame (other than the titlebar—i.e., the title area and command buttons) is designed to allow you to resize the window using the pointer. Notice that the frame is divided by small lines into eight sections: four long borders (two horizontal and two vertical) and four corners. Figure 3-9 shows these sections of the window frame.

You can resize a window horizontally, vertically, or simultaneously in both directions. Resizing is a bit trickier than any of the other window management functions we've tried, since the way you move the pointer determines the size of the window. It will probably take some practice.

If you place the pointer within a window and then move it into one of the long horizontal or vertical borders, you'll notice the pointer changes to a new shape: an arrow (pointing toward the window border), with a short line perpendicular to it. This short line represents the window border. Try moving the pointer in this fashion in one of the windows on your display to get a better idea of what the pointer looks like. If you move the pointer from within a window into the outer border at one of the corners, the pointer will become an arrow pointing diagonally at a small corner symbol, as pictured in Figure 3-10. Figure 3-11 shows all of the possible resize pointers.

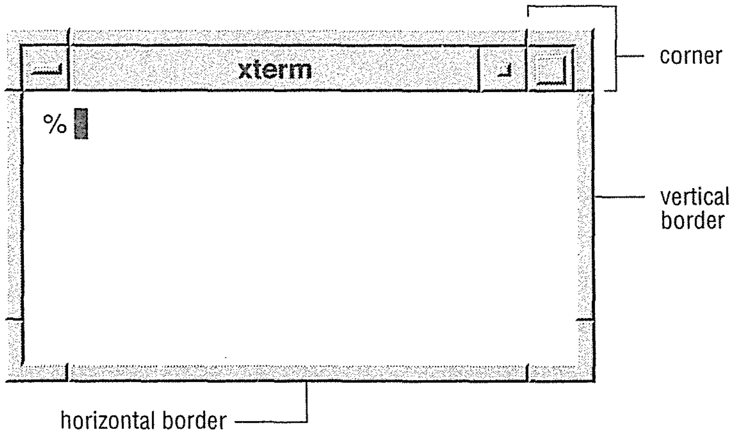


Figure 3-9. The outer frame is divided into four long borders and four corners

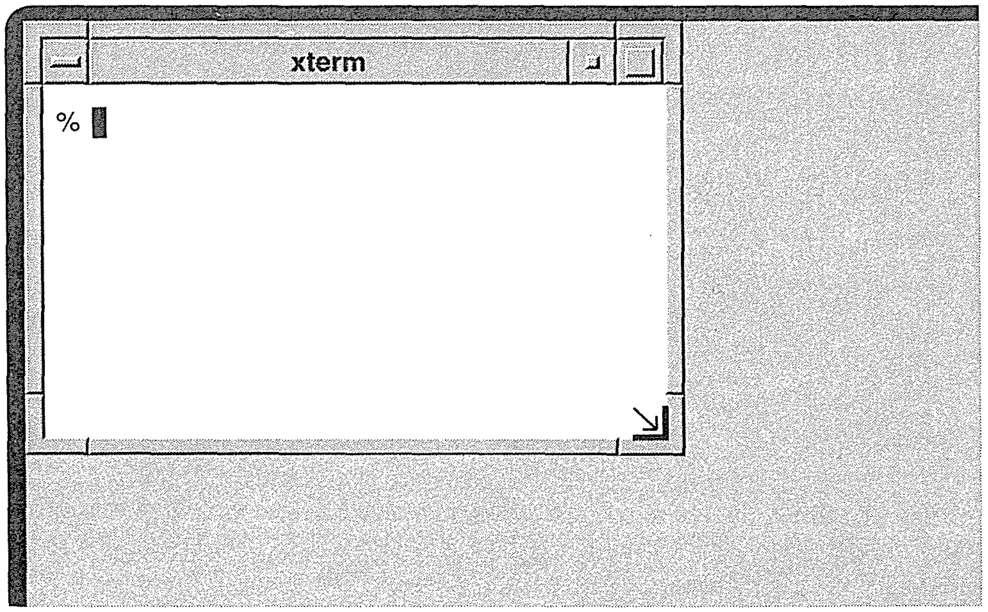


Figure 3-10. Window with resizing pointer

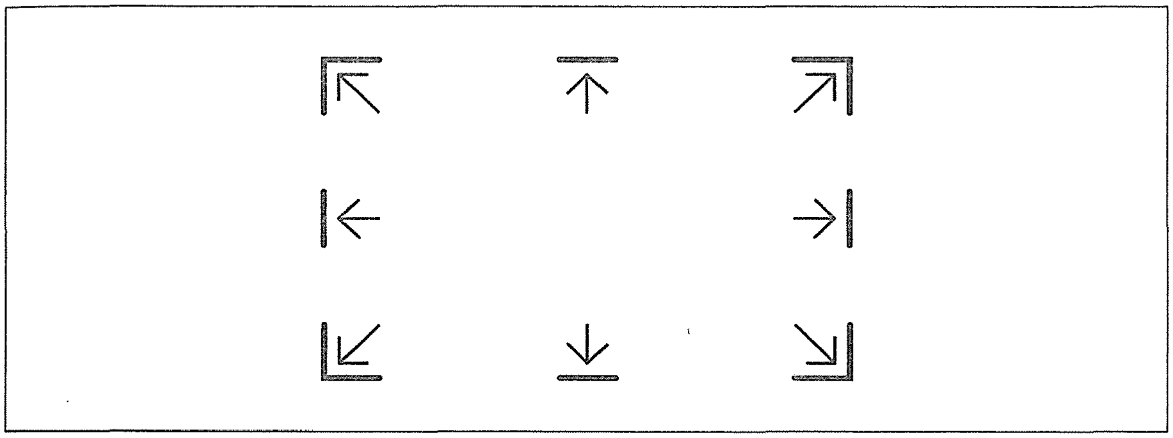


Figure 3-11. Resizing pointer symbols

Once the pointer changes to one of these shapes, you can move the border (or corner) of the window. Resizing from one of the long borders only allows you to change one dimension of the window: a horizontal border can only be moved up or down, changing the height; a vertical border can only be moved left or right, changing the width.

Resizing from a corner offers the most flexibility. You can move a corner in any direction you choose, changing both dimensions of the window if you want. For example, you can drag the lower-right corner of a window down and to the right to enlarge the window in both dimensions.

You determine the size and shape of the window by choosing the border or corner you want to extend (or contract) and moving it the desired amount using the following steps:

1. Move the pointer from within the window to the border or corner you want to move. The pointer changes to one of the symbols pictured in Figure 3-11.
2. Press and hold down the first pointer button and drag the window border or corner in the direction you want. As you resize the window, an image of the moving border(s) tracks the pointer movement. Also, in the center of the display, a small rectangular window shows the dimensions of the window as they change (in characters and lines for *xterm* windows, in pixels for most other clients).
3. Resize the window as desired.
4. Release the first pointer button. The window is redisplayed in the new shape. (The border image and window geometry tracking box disappear.)

Figure 3-12 shows a window being "stretched" from the lower-right corner.

Note that resizing an *xterm* window will not change the dimensions of the text currently in the window. If you make the window smaller, for instance, some of the text may be obscured. On most operating systems, this should not be a problem. As you continue to work, perhaps starting a text editor, the text will be adjusted to display in the newly sized window. Problems are more likely to occur if you resize a window *during* a text editing session. It's likely that the text editing program will not know about the window's new size and will operate incorrectly. To solve this problem, simply quit out of the editor and start another session.

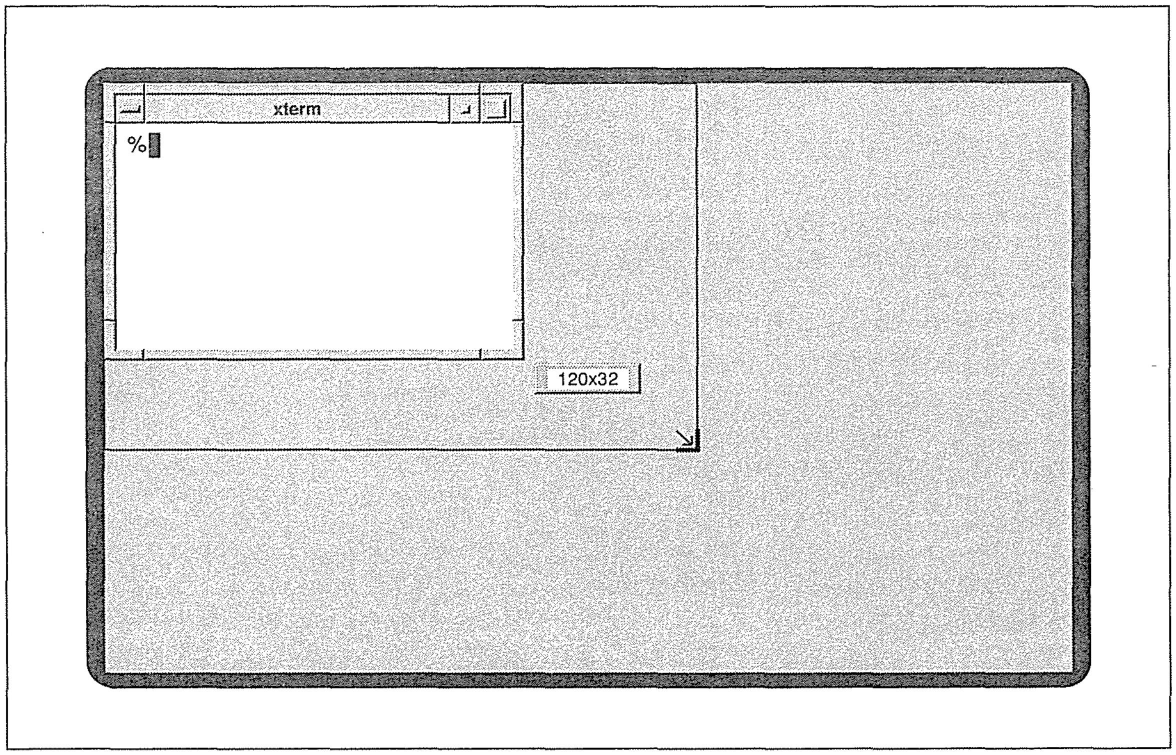


Figure 3-12. Dragging the corner to make a window larger

If your resized *xterm* window does not seem to know its new size, you may be working with an operating system that does not support terminal resizing capabilities. Refer to the discussion of the *resize* client in Chapter 5, *The xterm Terminal Emulator*, and to the *resize* reference page in Part Three of this guide for alternative solutions.

Closing a Window: The Window Menu Button

The command button on the left side of the titlebar is used to bring up the Window Menu, which provides seven items that can be used to manage the window and its icon. Chapter 4, *More about the mwm Window Manager*, describes how to bring up the Window Menu and invoke its various functions.

This command button also has another function. Double-clicking the first pointer button on the Window Menu command button kills the client program and closes the window. Be aware, however, that like other methods of "killing" a program (such as the *xkill* client), double-clicking on the Window Menu button can adversely affect underlying processes.

Refer to the section on *xkill* in Chapter 8, *Other Clients*, for a more complete discussion of the hazards of killing a client and a summary of alternatives.

You can customize *mwm* so that double clicking performs no function by setting a resource variable, `wMenuButtonClick2`, to false. See the sections "Setting mwm Resources" and "mwm-Specific Appearance and Behavior Resources" in Chapter 13, and the *mwm* reference page in Part Three of this guide for details.

Exiting from an xterm Window

When you are finished using an *xterm* window, you can remove it by typing whatever command you usually use to log off your system. Typically, this might be `exit` or Control-D. We'll close the second *xterm* window in Figure 3-13 by typing `exit`.

Notice that when we terminate the second *xterm* window, the first *xterm* window takes over the input focus. When explicit focus is being used and a window is terminated, the input focus reverts to the window that previously had the focus.

Be aware that terminating the login *xterm* window (the first *xterm* to appear) kills the X server and all associated clients. (If *xdm* is running X, the server will be reset but only after all client processes have been killed.) Be sure to terminate all other *xterm* windows before terminating the *xterm* login window. Also, if you are in an editor such as *vi*, be sure to save your data before you terminate the window.

In fact, it may be wise to iconify the login window and use other *xterm* windows instead, so that you don't inadvertently terminate it. Remember: you iconify a window by placing the pointer on the Minimize command button on the frame and clicking the first pointer button.

If you are worried about typing Control-D (end-of-file) accidentally and you normally use the C shell (*csh*), you can enter:

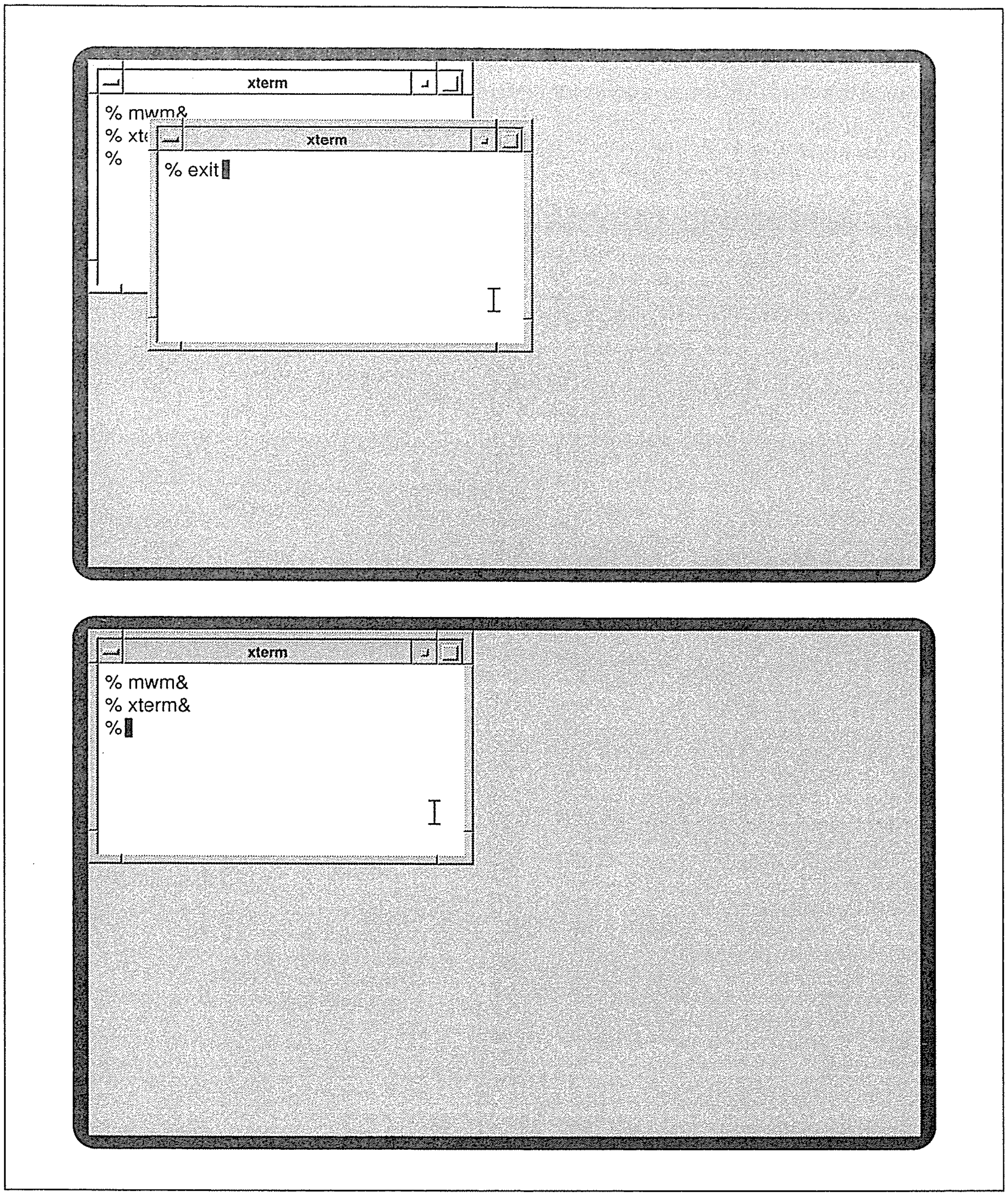
```
    % set ignoreeof
```

in the login window. Then typing `exit` becomes the only way you can terminate the window.

Note that some C shell implementations have an `autologout` variable, which will automatically terminate the shell if there is no activity for a given period of time. If your C shell supports this feature, be sure to disable it in the login *xterm* window using this command:

```
    % unset autologout
```

As an alternative to entering the command used to log off the system, you can also terminate an *xterm* window by selecting Send HUP Signal, Send TERM Signal, Send KILL Signal, or Quit from the *xterm* Main Options menu. (These menu options send different signals to the *xterm* process. Depending on what signals your operating system recognizes, some of the options may not work as intended. See Chapter 5 for more information.)



```
% mwm&
% xt
%
```

```
% exit
```

```
% mwm&
% xterm&
%
```

Figure 3-13. Closing an xterm window

The *mwm* window manager also provides several ways to remove an *xterm* or any other client window, among them double-clicking on the Window Menu command button, as described previously. Additional methods are described in Chapter 4, *More about the mwm Window Manager*.

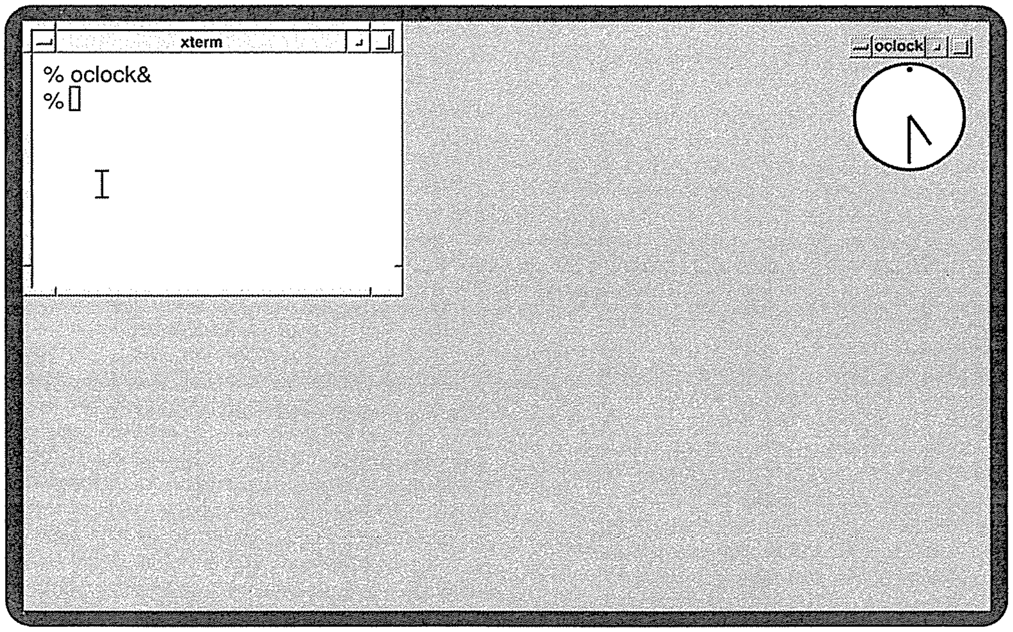
Starting Additional Clients

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

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

```
    % oclock &
```

you can start a clock application. The clock will appear in the upper-left quarter of the screen. With non-rectangular windows like *oclock*, a titlebar appears to float above the window. You can then drag the *oclock* window to a more convenient location—say the upper-right corner, as in Figure 3-14. (Remember, to move a window, place the pointer on the title area, press the first button, drag the window outline to the desired location, and release the button. Notice that the outline is rectangular, even if the *oclock* window isn't!)



```
% oclock&
%
```

Figure 3-14. The oclock window

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

Unfortunately the developers of *oclock* neglected to provide an easy way to remove it. For instructions on removing an *oclock* window, see Chapter 8, *Other Clients*.

Command-line Options

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

The next few sections illustrate some typical uses of these important options. In explaining how to use them, we introduce some new, perhaps somewhat involved concepts (such as the way distances can be measured on your screen). Bear with us. We feel that you need to master the `-geometry` and `-display` options in order to begin to take advantage of the powers of X.

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

Window Geometry: Specifying Size and Location

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

```
-geometry geometry
```

The `-geometry` option can be (and often is) abbreviated as `-g`, unless the client accepts another option that begins with "g."

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

```
widthxheight±xoff±yoff
```

Obviously, the first half of the string provides the *`width`* and *`height`* of the window. Many application windows are measured in pixels. However, application developers are encouraged to use units that are meaningful in terms of the application. For example, *xterm*'s dimensions are measured in columns and rows of font characters. More precisely, an *xterm* window is some number of characters wide by some number of lines high (80 characters wide by 24 lines high by default).

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

Many users may not be accustomed to thinking in terms of pixels. What exactly is a pixel? A pixel is the smallest element of a display surface that can be addressed by a program. You can think of a pixel as one of the tiny dots that make up a graphic image, such as that displayed by a terminal or a television. The number of dots (or pixels) per inch of screen determines the screen's *resolution*.* The more dots per inch, the higher the resolution (and, hypothetically, the sharper the picture).†

Since a pixel is a tiny unit of measurement, gauging sizes and distances in pixels will take some practice. For instance, what are the dimensions of your screen in pixels (its resolution)? The *xdpyinfo* client, described in Chapter 8, will tell you this; your workstation or X terminal documentation may also provide this information. *xdpyinfo* also tells you the screen's resolution in dots per inch.

Keep in mind that monitors can vary substantially. The Sun 19-inch monitor has dimensions of 1152 × 900 pixels and a resolution of 90 × 90 dots per inch (commonly abbreviated to dpi). The NCD 16-inch X terminal has dimensions of 1024 × 1024 pixels and a resolution of 106 × 106 dpi.

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

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

If you don't specify the x and y coordinates at which to place a client window, the client may provide a default location; if the application doesn't provide a default and *mwm* is running, the window manager will automatically position the window in the upper-left quarter of the screen.

For now, let's just specify a window's location and let the size be the application default. The x and y offsets can be either positive or negative. If you specify positive offsets, you're positioning the left side and top side of the window. Negative offsets are interpreted differently. The possible values for the x and y offsets and their effects are shown in Table 3-1.

*Hardware manufacturers generally equate resolution with the screen's overall dimensions in pixels. Thus, you also need to know the actual physical size of the monitor (in inches) to determine the dots per inch. We find the dpi figure more useful.

†There are other factors that determine the "picture quality" of a monitor, including "depth," or the number of bits per pixel. Depth relates to how many colors a monitor can display. See "How Many Colors are Available on My Screen?" in Chapter 12, *Specifying Color*, for more information.

Table 3-1. Geometry Specification x and y Offsets

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

For example, the command line:

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

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

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

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

If you want a window placed away from one or both edges of the screen, the guesswork starts. How many pixels away from the left side of the screen? How many pixels down from the top? You'll have to experiment with placing some clients on your screen to get a better idea of x and y offsets.

It's actually a good idea to start some windows and move them around on your screen using the pointer. While you're dragging a window, check the x and y offsets displayed in the small box *mwm* places in the center of the screen. These coordinates are the positive x and y offsets of the window (i.e., the offsets relative to the upper-left corner of the screen). This method for gauging location is fairly reliable.*

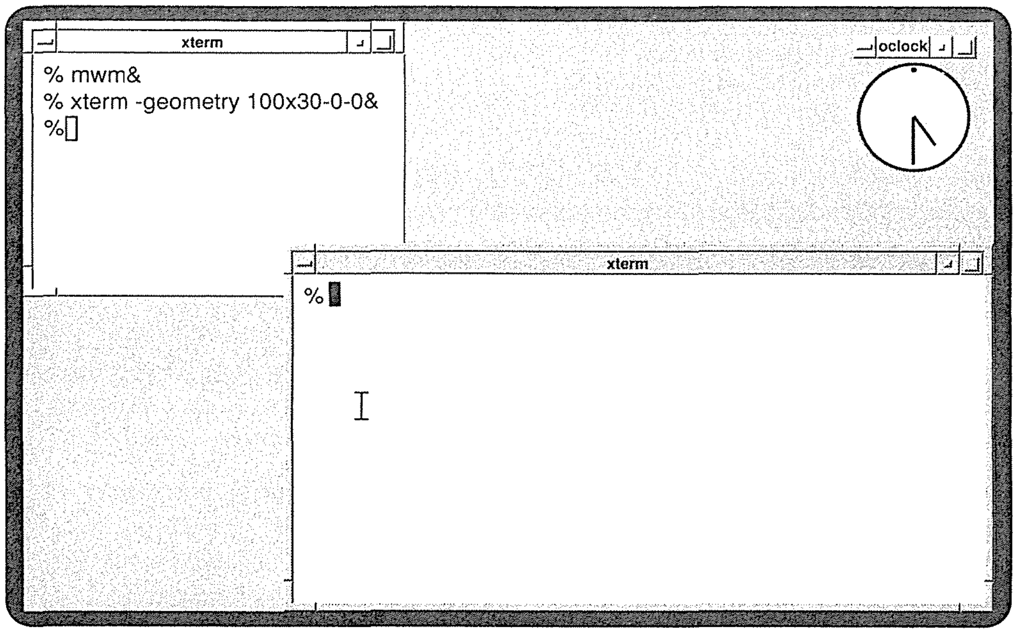
*There seems to be a very slight delay between pointer motion and update to the *mwm* coordinate box. When you finish dragging the window, the last coordinates visible in the box may differ from the true coordinates by a pixel in either or both directions. But this variance is so trivial that you can supply the coordinates as part of the geometry string and come very close.

You can also place some windows in different places by dragging and then determine their geometry specifications using the *xwininfo* client, described in Chapter 8, *Other Clients*.

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

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

This command creates a large *xterm* window in the lower-right corner of the screen, as illustrated by Figure 3-15.



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

```
%
```

Figure 3-15. xterm window sized and positioned with the -geometry option

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

Be aware that the geometry option is not necessarily the only means available to specify window size and location. Several clients, including *xterm*, allow you to set the size and location of a window (and often its icon or an alternate window) using resource variables (in an *.Xresources* or other defaults file). We'll introduce some of the basics of specifying resources later in this chapter. See Chapter 11, *Setting Resources*, for more detailed instructions. See the appropriate client reference pages in Part Three of this guide for a complete list of available resources.

You should be aware that, as with all user preferences, you may not always get exactly what you ask for. Clients are designed to work with a window manager, which may have its own rules for window or icon size and placement. However, priority is always given to specific user requests, so you won't often be surprised.

Running a Client on Another Machine: Specifying the Display

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

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

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

In order to run a client remotely and display its results locally, you must tell the client process where to display its window. For this purpose, X provides a command-line option: `-display`. Think of `-display` as a pointer to the physical display on which you want the window to appear. Like `-geometry`, the `-display` option is recognized by most X clients. The display option tells the client on which server to display results (i.e., create its window). The option has the syntax:

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

The `-display` option can be abbreviated as `-d`, unless the client accepts another option that begins with "d."

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

In this context, "host" refers to the Internet address name of the display hardware. It might be the name of a single-user workstation, an X terminal, a PC running an X server, or another hardware device.

"Server" refers to an instance of the X server program, which controls a single physical display. An X display may be composed of multiple screens, but the screens share one keyboard and pointer. Most workstations have only one keyboard and pointer and thus are classified as having only one display. Multiuser systems may have multiple independent displays, each running a server program. If one display exists, as in the case of most workstations and X terminals, it is numbered 0; if a machine has several displays, each is assigned a number (beginning with 0) when the X server for that display is started.

Similarly, if a single display is composed of multiple screens (sharing one keyboard and pointer), each screen is assigned a number (beginning with 0) when the server for that display is started. Multiple screen displays may be composed of two or more physical monitors. Alternatively, two screens might be defined as different ways of using the same physical monitor. For example, on our Sun-3/60 color workstation, screen 0 is color, and screen 1 is monochrome.* Each screen is the size of the monitor; you can only view one screen at a time. In practice, the two screens seem to be side by side: you can "scroll" between them by moving the pointer off either horizontal edge of the screen.

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

Although much of the current X Window System documentation suggests that any of the parameters to the `-display` option can be omitted and will default to the local node, server and screen 0, respectively, we have not found this to be true. In our experience, only the *host* and *screen* parameters (and the period preceding *screen*) can be omitted. The colon and *server* are necessary in all circumstances.

The DISPLAY Environment Variable

Technically speaking, the `-display` option allows you to override the contents of the DISPLAY environment variable, which stores the display name on UNIX systems. On UNIX systems, the display name is stored in the DISPLAY environment variable. Clients running on the local system access this variable to determine which physical display to connect to (for most clients, "connecting" is equivalent to opening a window). The DISPLAY variable is set automatically by *xinit* or *xdm* when you log in and the *xterm* terminal emulator inherits it. Thus it is set when you start X and conveyed to the first *xterm* window.

For most single-user systems, such as workstations, the X startup program sets the server and screen numbers to 0 and omits a hostname (the local host is assumed). (Prior to Release 5, in some cases the startup program sets the hostname to "unix," a generic name, which also defaults to the local host.) If you are working on a single-user system and run all processes on it, you don't have to deal with setting the display. Clients running locally access the DISPLAY variable and open windows on a display connected to the local host.

If you are using an X terminal, it should already be configured so that the DISPLAY variable is set properly when you log on to the local host. Again, if you run all process on the local machine (to which your terminal is connected), you don't have to deal with specifying the display. Clients will access the DISPLAY variable and open windows on your X terminal screen.

* See Volume Eight, *X Window System Administrator's Guide*, to learn how to configure your Sun workstation to use dual frame buffers.

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

Complications arise when you want to run a process on a remote machine and display the results locally. A client running on a remote machine does not have access to the DISPLAY variable, which is stored in the local shell. You can use the `-display` option to specify which display the client should access.

Using -display

Suppose you're using a single display workstation and the display also has only one screen. The hostname of the workstation is *kansas*. In order to tell a client to connect to a display, you must identify it by its unique name on the network. (You cannot identify your display by the shorthand setting given to it by the X startup program—`unix:0.0`, `:0.0` or some variation.) Let's assume that the complete display name for the workstation *kansas* is:

```
kansas:0.0
```

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

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

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

Keep in mind that for this process to succeed, the remote client (running on *oz*) must have permission to "open" the local display (on *kansas*). (See "Server Access Control" in Appendix A, *Managing Your Environment*, and the *xhost* reference page in Part Three of this guide, for more information.) If *oz* has not been granted access to the server running on *kansas*, the window will not be opened, and you may also get an error message similar to:

```
Error: Can't Open display
```

If your command fails, try entering the command:

```
% xhost +
```

in an *xterm* window running on your display. Then run the remote shell (*rsh*) again. If that works, consult your system administrator or Volume Eight, *X Window System Administrator's Guide*, to learn how to set up access control properly.

The following command illustrates a common mistake:

```
% rsh oz xterm &
```

*The command to run the remote process might be different depending on the available networking software. Ask your system administrator for the proper command.

If you try to run a client using a remote shell and forget to direct the client to create its window on your own display, the window will not be displayed and you'll get an error message stating that the display cannot be opened.

If you're using an X terminal, it will have a name unique to the terminal (e.g., ncd8), which should be used as the `host` component of the `-display` argument. You should not use the name of the system to which the terminal is connected.

Suppose we have a terminal with the full display name:

```
    ncd8:0.0
```

(connected to *kansas*) and we want to run an *xterm* on *oz*. The command:

```
    % rsh oz xterm -display ncd8:0.0 &
```

will open the window on our X terminal.

In addition to specifying a local display, if permissions allow you can also use the display option to open a window on someone else's display. You might want to display a window on another user's screen for instructional purposes. Multiuser systems can even be set up to allow teachers to display educational material simultaneously to several students, each using an X display of some sort. And, of course, you might want to display a window on a friend's screen, just for the fun of it. (The security problems that allow both innocent pranks like this, as well as more serious breaches, are described in Volume Eight, *X Window System Administrator's Guide*.)

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

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

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

Once You Run a Remote xterm Using –display

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

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

```
    % echo $DISPLAY
```

The system should echo:

```
kansas:0.0
```

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

Logging In to a Remote System

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

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

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

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

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

Monitoring the Load on a Remote System

A client you may wish to run on another machine is *xload*, which is used to keep track of the system load average. By default, *xload* polls the system for the load average at ten-second intervals and displays the results in a simple histogram.

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

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

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

```
% xload &
```

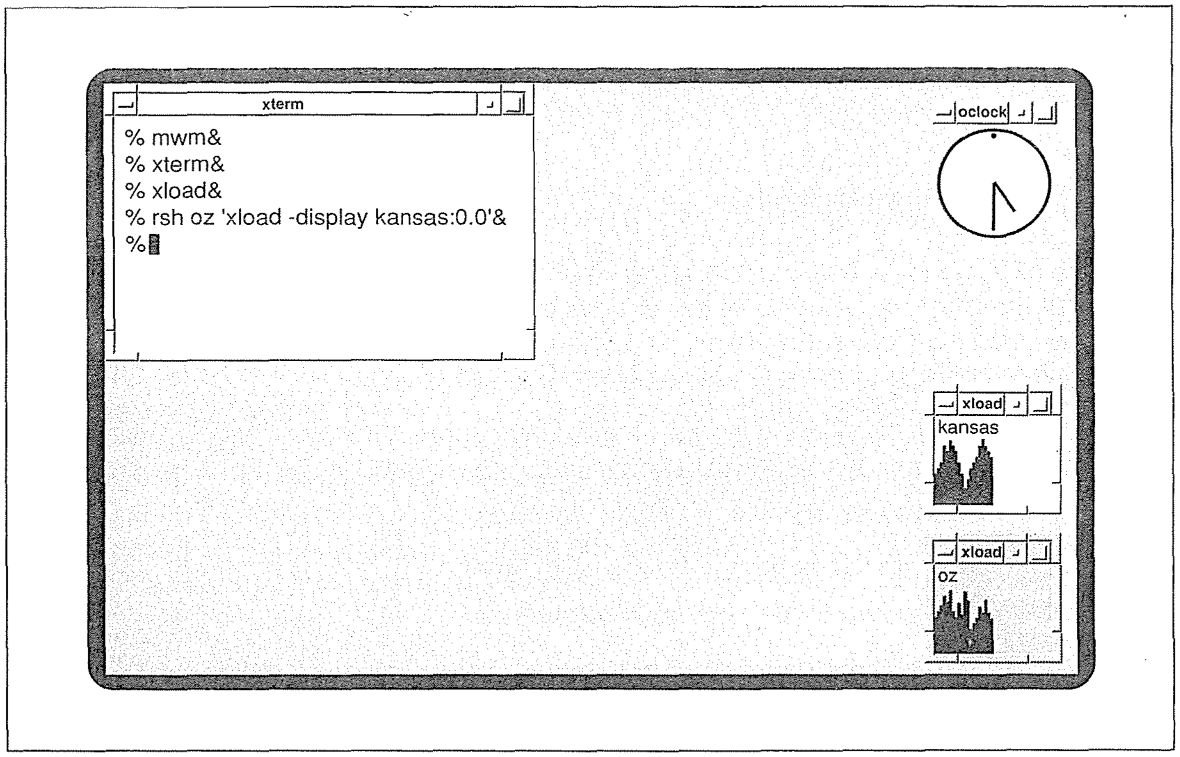
Once the *xload* window is created, move it to a convenient location on the screen.

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

```
% rsh oz xload -display kansas:0.0 &
```

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

Figure 3-16 shows the resulting *kansas* display: two *xload* windows—the top window monitoring activity on the local system and the bottom one monitoring activity on the remote system.



```
% mwm&
% xterm&
% xload&
% rsh oz 'xload -display kansas:0.0'&
%
```

Figure 3-16. Monitoring activity on two systems with xload

Putting It All Together

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

Suppose we're using a Sun 3/60 workstation with the hostname *jersey*. The workstation has a single display with two screens: screen 0 is color and screen 1 is black and white. Once X and the window manager* are running, we might set up the display using the following commands.

*Note that you should run *mwm* with the `-multiscreens` option to have the program manage all screens on the display. If you start *mwm* without this option, it only manages screen 0. In this case, you can either kill and restart it with `-multiscreens` or run another instance of it on screen 1.

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

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

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

```
% xload -geometry -10-200 &
% rsh manhattan xload -geometry -10-20 -display jersey:0.0 &
```

Run another *xterm* window on *jersey*.

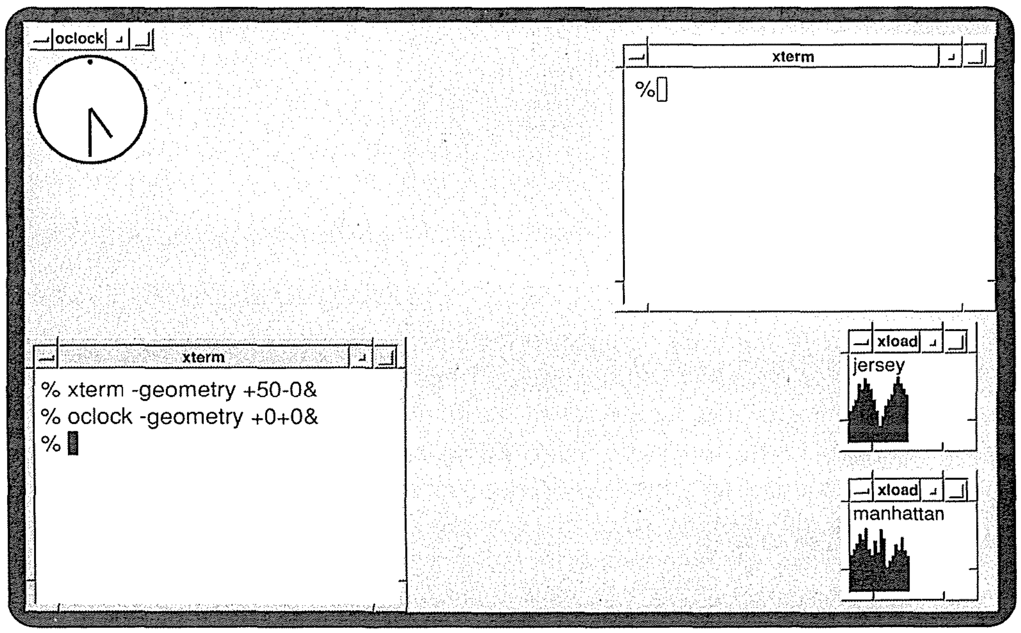
```
% xterm -geometry +50-0 &
```

Then iconify the login *xterm* window so that you don't inadvertently kill it (and shut down X in the bargain). Remember: to iconify a window place the pointer on the Minimize command button on the window's frame and click the first button. (See "Converting a Window to an Icon" earlier in this chapter.)

Run an *oclock*.

```
% oclock -geometry +5+5 &
```

Figure 3-17 shows the *jersey* display, screen 0: a fairly useful layout.



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

Figure 3-17. A working display, screen 0

You might also place client windows on the workstation's alternate screen, the black and white screen numbered 1. By default, windows are always placed on screen 0 but you can place a client window on screen 1 by specifying the screen number in the `-display` option when starting the client. For instance, each of the following commands places an *xterm* window on screen 1.

```
% xterm -display jersey:0.1 &

% rsh manhattan xterm -geometry -0-0 -display jersey:0.1 &
```

Figure 3-18 illustrates screen 1.

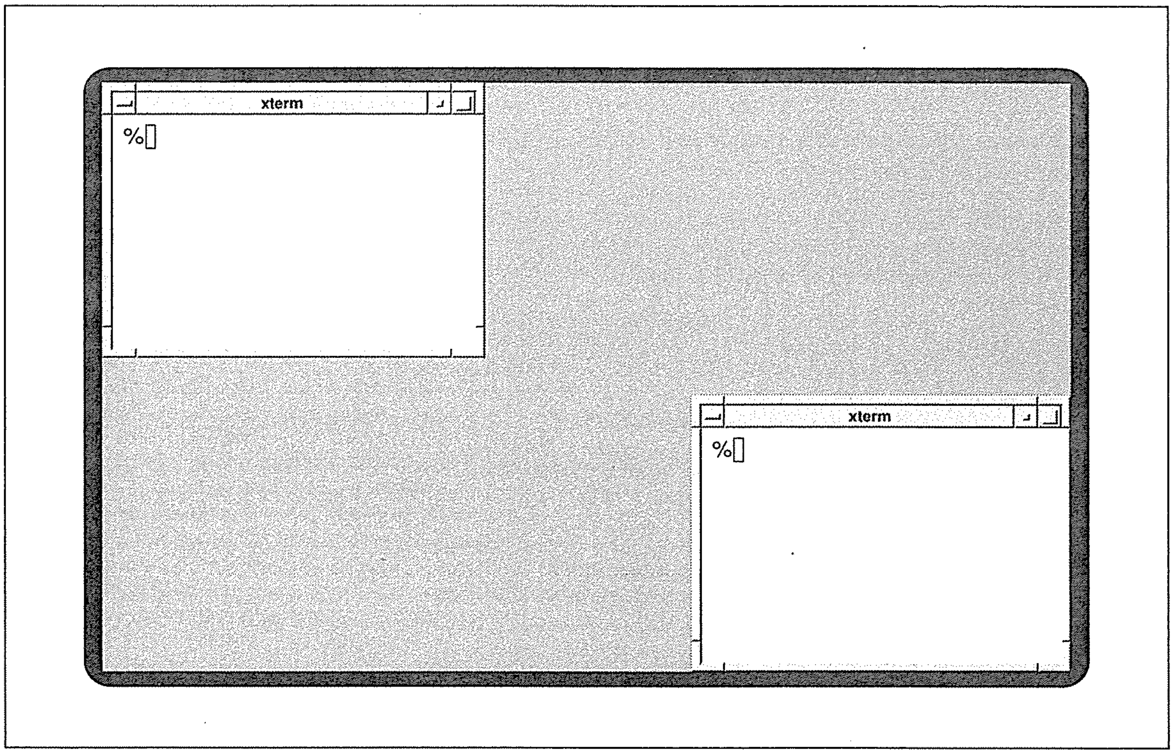


Figure 3-18. A working display, screen 1

As we'll see in Appendix A, *Managing Your Environment*, on most systems you can place the commands you run to set up your display in a special file that is invoked when you log in. Once this file (usually called either *.xinitrc* or *.xsession*) is in place, when you log in your display will be set up to your specifications automatically.

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

Customizing a Program

In a sense, command-line options allow you to customize one program. We've already seen how to use the `-geometry` and `-display` options, which are accepted by most clients. Chapter 10, *Command-line Options*, describes some of the other options accepted by most of the standard clients. These options set window features such as:

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

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

Like most clients, *xclock* accepts a variety of options. Some of *xclock*'s options are intended to enhance the clock display aesthetically and some to affect its operation. Taking a look at a few of *xclock*'s options should give you a better idea of the flexibility of X.

The following command line runs a custom *xclock* display:

```
% xclock -hd green -hl royalblue -bg lightblue -fg royalblue -update 1 -chime &
```

As you can see, these specifications are intended for a color monitor. (X is highly flexible in the use of color. See Chapter 12, *Specifying Color*, for more information.) The `-hd` option sets the color of the clock's hands to green. The `-hl` option provides even more detail, specifying the color of the outline of the hands as royal blue. `-bg` and `-fg` are two of the options accepted by most clients; they set the window's background and foreground color, in this case to light blue and royal blue, respectively.

The `-update` option takes as its argument the frequency in seconds at which the time on the clock should be updated. We've specified that the time be updated at one second intervals. Thus a second hand (also green) will be added to the *xclock* display. (The *xclock* reference page in Part Three of this guide specifies that a second hand is added if `-update` is given an argument of less than 30 seconds.)

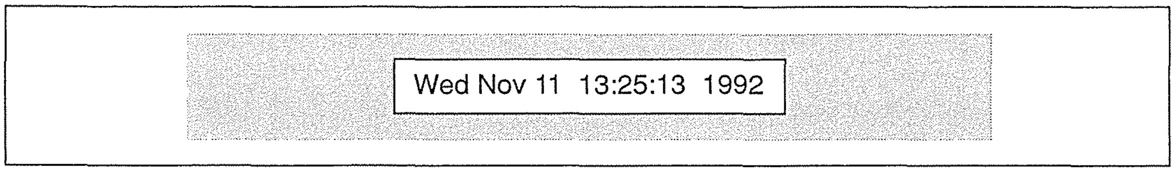
The `-chime` option specifies that the keyboard bell will ring once on the half hour and twice on the hour.

These options create a somewhat fancy *xclock* display. You might or might not want to use so many options, but these and several more are available.

By default, *xclock* displays a traditional clock face (an analog clock). You can create a digital *xclock* using the following option:

```
    % xclock -digital &
```

The digital *xclock* is pictured in Figure 3-19.



Wed Nov 11 13:25:13 1992

Figure 3-19. Digital xclock display

Logically, the `-hd` and `-hl` options, which set the color of the clock hands, are only valid with the analog (default) *xclock*. For a complete list of options, see the *xclock* reference page in Part Three of this guide.

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

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

X offers an alternative to customizing a single client process on the command line. You can specify default characteristics for a client using variables called *resources*.

Customizing the X Environment

Command-line options allow you to customize one instance of a client program. In addition, X provides a mechanism that allows you to specify characteristics that take effect *every* time you run a client. Almost every feature of a client program can be controlled using a variable called a *resource*. You can change the behavior or appearance of a program by changing the *value* associated with a resource variable. (In some cases, a resource variable controls the same characteristic as a command-line option. However, while the option specifies a characteristic for the single client process being invoked, a resource variable makes the characteristic the program default.)

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

By default, the way a client looks and behaves is determined by the program code, and in some cases, by a system-wide file of *application defaults*. Several clients have application defaults files that determine certain client features.* Within an application defaults file, defaults are set using resources. The resources specified in a client's application defaults files are usually just a subset of a greater number of resources that can be set.

If the characteristics you set in your own resources file already have system-wide application defaults, your own settings take precedence. Keep in mind, however, that command-line options override both your own defaults and any system-wide defaults for the single client process.

To make your resource specifications available to all clients, X provides a program called *xrdb*, the X resource database manager. *xrdb* stores resources directly in the server where they are accessible to all clients, regardless of the machine the clients are running on.

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

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

```
    client*variable:    value
```

The following example shows five simple resource specifications for the *xclock* client. These particular resources specify the same characteristics as the command-line options we used to create the green and blue *xclock* in the preceding section.

Example 3-1. Resources to create a custom xclock

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

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

1. In your home directory, create a file containing the resources listed in Example 3-1. Name the file *.Xresources*. (A resource file can actually have any name, but is often called *.Xresources* or *.Xdefaults*.)

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

Application defaults files generally reside in the directory */usr/lib/X11/app-defaults* and are named for the client application. In describing the appearance and behavior of clients in this guide, we assume all of the standard application defaults files are present on your system and accessible by the client programs. If, by some chance, a client's application defaults file has been edited or removed from your system, the client may not look or behave exactly as we describe it. If a client application appears substantially different than depicted in this guide, you may be using a different version of the program or the application defaults may be different. Consult your system administrator.

2. Load the resources into the server by entering the following command in an *xterm* window:

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

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

You should load resources using *xrdb* every time you log in. In Appendix A, *Managing Your Environment*, we'll describe how to automate this process using a special startup script, which also opens the client windows you want on your display.

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

Resource specifications can be much more complicated than our samples suggest. For applications written with a toolkit (such as the X Toolkit or the Open Software Foundation's Motif Toolkit), X allows you to specify different characteristics for individual components, or *widgets*, within the application. Typical widgets create graphical features such as menus, command buttons, dialog boxes, and scrollbars. Within most toolkit applications is a fairly complex widget hierarchy—widgets exist within widgets (e.g., a command button within a dialog box).

Resource naming syntax can parallel the widget hierarchy within an application. For instance, you might set different background colors for different command buttons and specify still another background color for the dialog box that encloses them. In such cases, the actual widget names are used within the resource specification. Chapter 11, *Setting Resources*, explains the resource naming syntax in greater detail and outlines the rules governing the precedence of resources. It also explains how to use the *editres* program to examine a (standard) client's widget hierarchy and set resources accordingly.

Where to Go from Here

There are many useful client programs supplied with the X Window System. Details of how to use one of the most important clients, the *xterm* terminal emulator, are provided in Chapter 5. Clients to list and display fonts are described in Chapter 6, *Font Specification*. Chapter 6 also describes the X font naming conventions and various ways to specify fonts on the command line (and in resource files). Chapter 7 describes several *Graphics Utilities* available with X. An overview and tutorial for other standard clients and instructions on using certain public domain clients are provided in Chapter 8, *Other Clients*. Chapter 9, *Working with Motif Applications*, gives instructions on using Motif applications. All clients are described in detail in a reference page format in Part Three of this guide.

We've introduced some basic operations you can perform using the *mwm* window manager. For instructions on performing additional window manager operations, such as lowering a window, read Chapter 4, *More about the mwm Window Manager*. You can then go on to read more about *xterm* in Chapter 5 and about some of the other standard clients in Chapters 6 through 8.

4

More About the mwm Window Manager

This chapter describes additional functions you can perform using the Motif window manager, mwm.

In This Chapter:

4
More About the mwm Window Manager

Chapter 2, *Getting Started*, describes how to tell if *mwm* is running on your display and how to start it if you need to. Chapter 3, *Working in the X Environment*, describes some of *mwm*'s most basic and useful capabilities, which you can perform using the pointer on a window's frame. It explains how to iconify, maximize, raise, move, resize, and close windows, how to move icons, and how to convert icons back to windows.

The current chapter describes how to perform some of these window management functions in alternative ways and also describes some additional functions, including:

- Creating additional *xterm* windows.
- Lowering windows (moving them to the back of others).
- Shuffling windows on the display.
- Refreshing your screen.
- Restarting the window manager.

The Motif window manager allows you to invoke window management functions in a variety of ways:

- Using the window "frame" and various features available on it: the Minimize (iconify) button, Maximize button, title area, Window Menu, etc.
- Using the Root Menu.
- Using keyboard keys, pointer buttons, and key and button combinations.

In this chapter, we'll review some basics about focusing input to a window or icon and consider how window management functions rely on focus being directed properly. Then we'll take a closer look at the Window Menu and Root Menu.

As we'll see, some window management functions can be performed using keyboard shortcuts. These shortcuts involve using special keys, commonly called modifier keys. Before considering window management functions, we'll take a brief look at the special keys significant to X.

Keep in mind that *mwm* is very flexible. In Chapter 13, we'll consider how to customize various features of the window manager. Perhaps the most useful customization that can be performed involves selecting a keyboard focus policy, either pointer focus or click-to-type (also referred to as *explicit*) focus. By default, *mwm* uses click-to-type focus. Keyboard focus is described in Chapter 1.

The current chapter is intended primarily for those using Version 1.2 of *mwm*. (Unlike the 1.1 version, *mwm* 1.2 is compatible with Release 5 of X.) If *mwm* has been customized at your site or you are running a different version, the principles should be basically the same, but the window management functions may be invoked in different ways. From time to time, we'll mention how commands or functionality might vary, depending on the version of *mwm*.

Using Special Keys

Undoubtedly you know the basics of using a keyboard. However, X interprets certain keys somewhat differently than the labels on the keys would indicate.

Most workstations have a number of "modifier" keys, so-called because they modify the action of other keys. Generally these keys are used to invoke commands of some sort, such as window manager functions.

Three of these modifier keys should be familiar to any user of a standard ASCII terminal or a personal computer—Shift, Caps Lock, and Control. However, many workstations have additional modifier keys as well. A PC has an "Alt" key, a Macintosh™ has a "fan" key, a Sony workstation has keys named "Nfer" and "Xfer," and certain model Sun workstations have three additional modifier keys, labeled "Left," "Right," and "Alternate."

Because X clients are designed to run on many different workstations, with different keyboards, it is difficult to assign functions to special keys on the keyboard. A developer can't count on the same key always being present!

For this reason, many X clients make use of "logical" modifier keynames, which can be mapped by the user to any actual key on the keyboard.

Up to eight separate modifier keys can be defined. The most commonly used (after Shift, Caps Lock, and Control) has the logical keyname "Meta."

We'll talk at length about this subject in Chapter 14, *Setup Clients*, but we wanted to warn you here. When we talk later in this chapter about pressing the "Meta" key, you should be aware that there is not likely to be a physical key on the keyboard with that name. For example, on one workstation, the Meta key might be labeled "Alt" and, on another, "Funct." And as we'll show in Chapter 14, you can choose any key you want to act as the Meta key.

Unfortunately, X provides no easy way to find out which key on your keyboard has been assigned to be the Meta key. When you need to know, please turn to the discussion of key mapping in Chapter 14, for information on how you can find out.

Input Focus and the Window Manager

As explained in Chapter 3, the default operation of *mwm* is that you select the window to receive input (the active window) by clicking the first pointer button anywhere within the window. This focus policy is called click-to-type, or explicit. Whether *mwm* is started automatically or you started it by typing in an *xterm* window, you must then click in a window in order to enter text.

Once you focus input to a window, all text typed is interpreted by that client, regardless of where you move the pointer. In the case of *xterm* and similar clients, the text appears in the window. Other clients might simply interpret the keystrokes as possible commands. Some clients, such as *xclock*, do not recognize keyboard input at all. Regardless of the client, certain keystrokes might also be interpreted as commands to the window manager to effect changes on the focus window. In order to type in or issue commands to affect another window, you must transfer focus to that window by clicking the first pointer button within it. In Chapter 13, *Customizing mwm*, we'll describe how to make the keyboard focus follow pointer movement.

Also keep in mind that in order to manage a window using the various methods provided by *mwm*, the window must have the input focus. Since most actions you'll perform involve placing the pointer somewhere on the frame and performing some kind of pointer action (such as clicking, pressing, releasing, etc.), controlling a window using the frame selects that window to receive the input focus at the same time.

However, before invoking certain window management functions, you must first select a focus window (or icon). For example, each Window Menu item has a keyboard shortcut. If you want to affect a window using a keyboard shortcut, you must *first* select it as the focus window by clicking on it with the pointer—otherwise the keyboard shortcut will affect the current focus window. This is probably the only circumstance in which you would select a window that doesn't accept keyboard input (such as *xclock*) as the focus window. Keyboard shortcuts for Window Menu functions are described in "Using the Window Menu."

As we've seen, when you focus input on a window, the window frame changes color. The color of the active window's frame depends on the version of *mwm* you are running and the color resources for your system. In some versions, be aware that the black window frame of non-active windows obscures the titlebar text, which also appears in black. Only the title of the active window is visible in these cases.

If you are working with a stack of windows that overlap, selecting a window as the active window automatically raises that window to the top of the stack. This behavior is controlled by an *mwm* resource variable called `focusAutoRaise`, which is true by default when click-to-type focus is in effect. (This resource is described in Chapter 13, *Customizing mwm.*) If the focus window is killed or converted to an icon, the focus switches back to the window that previously had it.

Focusing Input on an Icon

Though it may not be immediately obvious, you can select an iconified window to receive the input focus. You direct input to an icon by placing the pointer on it and clicking the first button. As illustrated in Chapter 3, *Working in the X Environment*, this action displays an *mwm* menu (the Window Menu) over the icon. (See Figure 3-4.)

The icon label also becomes wider and the label and the frame surrounding the icon are highlighted. These changes indicate that the icon has the input focus. Even after the menu is removed, the label will remain wider and the border highlighted (that is, the icon will retain the focus), until you direct focus to another window or icon.

When an icon has the input focus, the application window the icon represents will not interpret keystrokes. However, the window manager will interpret relevant keystrokes as commands. For instance, you need to focus input to an icon in order to invoke a Window Menu item on it using the item's keyboard shortcut. Keyboard shortcuts for Window Menu functions are described in "Using the Window Menu on Icons" later in this chapter.

Changing the Stacking Order with Keystrokes

Regardless of the focus policy in effect (either pointer focus or the default click-to-type), you can shuffle windows on the display using the following key combinations.

Table 4-1. Key Combinations to Change the Stacking Order

Key Combination	Action
Meta-Escape	Move top window to bottom of stack.
Meta-Shift-Escape	Move bottom window to top of stack.

The only difference between these keystroke combinations is the direction in which they reorder the windows.

The key combination Meta-Escape moves the window currently at the top of the stack to the bottom. (The second highest window becomes the top window.)

The key combination Meta-Shift-Escape moves the window currently at the top and given the input focus. (The second lowest window becomes the bottom window.)

As explained previously, there is probably no key labeled "Meta" on your keyboard. Meta is a logical keyname recognized by X which is mapped to a physical key. Also be aware that, in most cases, *mwm* considers the Meta key and the Alt key interchangeable. (Alt is another logical keyname that may be mapped to a key with another label.) Thus, the actions in Table 4-1 should also work if you substitute Alt for Meta. For help in locating the Meta and Alt keys on your keyboard, see the discussion of *xmodmap* in Chapter 14, *Setup Clients*.

Note that changing the stacking order in this way does not affect which window has the input focus. If you're using click-to-type (i.e., explicit) focus, you can also transfer focus using keystrokes, as we'll see in the next section.

Perhaps somewhat less relevant to the average user is a client's "local" stacking order. Since many client applications provide transient subwindows (dialog boxes, help windows, etc.), a client itself can be thought to have a stacking order. By default, when you raise or lower a window, all associated subwindows are moved with the primary application window and remain on top of it. However, version 1.2 of *mwm* provides mechanisms to affect how the local stacking order is affected when the main window is shuffled. For more information, see the "Actions" `f.lower`, `f.raise`, and `f.raise_lower` on the *mwm* reference page in Part Three of this guide.

Transferring (Explicit) Focus with Keystrokes

If you are running the default version of *mwm* (which assumes click-to-type focus and `focusAutoRaise`), you can cause the focus to circulate from window to window (including iconified windows) within the stack using any of the key combinations that appear in Table 4-2.

Table 4-2. Key Combinations to Change Focus Window

Key Combination	Action
Meta-Tab	Move focus to next window in stack.
Meta-Shift-Tab	Move focus to previous window in stack.

The key combination Meta-Tab circulates the focus from the current top of the stack to the bottom. What this means is that the top window in the stack is moved to the bottom; the second highest window becomes the top window and gets the input focus.

The key combination Meta-Shift-Tab circulates the focus from the current bottom of the stack to the top. In other words, the lowest window in the stack is moved to the top and given the input focus.

Again, note that the key that performs the "Meta" function varies from keyboard to keyboard. Remember also that in most cases, *mwm* considers the Meta key and the Alt key interchangeable. Thus, the actions in Table 4-2 should also work if you substitute Alt for Meta. For help in locating the Meta and Alt keys on your keyboard, see the discussion of *xmodmap* in Chapter 14, *Setup Clients*.

mwm's ability to transfer focus from window to window according to keystrokes is important to some Motif applications. The Motif Toolkit supports the building of applications that use several subwindows, each with a different, albeit related, purpose. (Such applications can be described as *form-based*.) A form-based Motif application can be built that allows users to move among the subwindows using keystrokes.

Some of the more common elements of Motif applications are described in Chapter 9, *Working with Motif Applications*.

What To Do if mwm Dies and the Focus Is Lost

If the *mwm* process dies, the focus policy should revert to pointer focus. To restart the window manager, you should simply be able to move the pointer into an *xterm* window and enter the *mwm* command. However, *mwm* has a bug that makes it possible to lose the focus entirely when the window manager dies. Obviously, restarting the window manager in such a situation is problematic. If *mwm* dies and no window retains the focus, you can restore focus to an *xterm* window using the Secure Keyboard item of the client's Main Options menu. See Chapter 5, *The xterm Terminal Emulator*, for details.

A Quick Review of Frame Features

If you're comfortable with managing windows using the frame, skip this section and go on to "Using the Window Menu."

When *mwm* starts up, it places a frame around every window on the screen. The frame has several components: a titlebar, featuring the name of the application (in the *title area*) and three command buttons (Maximize, Minimize, and Window Menu); and an outer border that permits resizing of the window.

In Chapter 3, *Working in the X Environment*, we demonstrated the window management functions you can perform using the pointer on the *mwm* frame, which appears surrounding an *xterm* window in Figure 4-1.

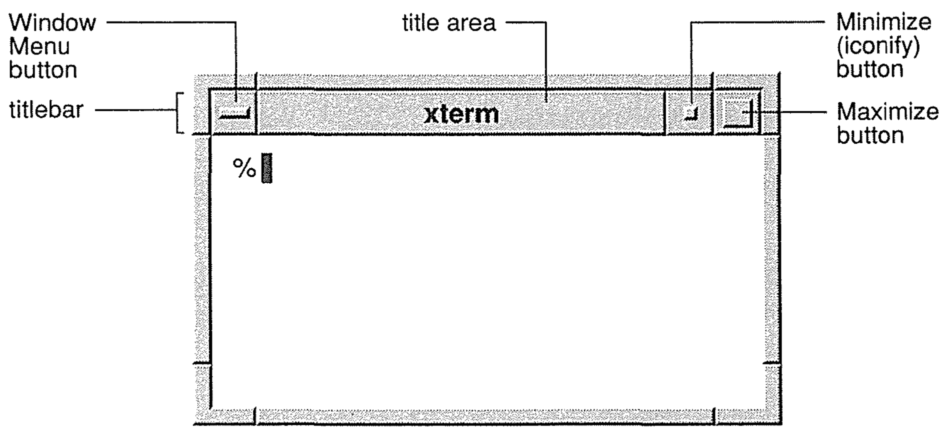


Figure 4-1. An xterm window running with the Motif window manager

Using the pointer on various parts of the frame, you can:

- Iconify the window, by clicking the first pointer button on the Minimize command button, the lefthand button in the upper-right corner of the frame (decorated with the smaller square). To convert an icon back to a window, double-click on the icon with the first pointer button.
- Maximize the window (convert it to the maximum size allowed by the client, often root window size), by clicking the first pointer button on the Maximize command button. The Maximize button is to the right of the Minimize button and is decorated with the larger square. To convert the window back to its original size, click on the Maximize button again.
- Raise a window, by clicking the first pointer button on any part of the window frame, except the three command buttons (Minimize, Maximize, and Window Menu). To raise an icon, click the first pointer button on it. This action also posts the Window Menu over the icon. (Remove the menu by clicking somewhere off the menu.) When you are using explicit (click-to-type) focus, raising also focuses input.
- Move the window, by pressing and holding down the first pointer button on any part of the title area (except the command buttons), dragging the window outline to a new location, and releasing the pointer button. (A small box in the center of the screen displays the changing coordinates.) To move an icon, place the pointer on the icon, press and hold down the first pointer button, drag the icon outline to a new location, and release the pointer button.
- Resize the window, by moving the pointer into one of the resize borders or corners, pressing and holding the first pointer button, and dragging the border or corner in the direction you want. A window outline tracks the resize operation. Release the pointer button to redraw the window in the selected dimensions.
- Close the window, by double-clicking the first pointer button on the Window Menu command button (in the upper-left corner of the frame). See the section on *xkill* in Chapter 8, *Other Clients*, for a discussion of the liabilities of "killing" a client.

All of these items can also be invoked using Window Menu items or keyboard shortcuts for these items. The Window Menu also provides an additional item to Lower a window or icon. Let's take a look at the Window Menu, see how to display it, and consider the various ways to invoke its functions. Then we'll look at pointer commands and menu items to manage icons. Finally, we'll consider those functions available on *mwm*'s Root Menu.

Using the Window Menu

The command button on the left side of the titlebar is used to bring up the Window Menu, which provides seven items that can be used to manage the window and its icon. The next few sections describe how to bring up the Window Menu and invoke its various functions.

As we saw in Chapter 3, this command button also has another function. Double-clicking the first pointer button on the Window Menu command button kills the client program and closes the window. See *xkill* in Chapter 8, *Other Clients*, for a discussion of the hazards of this action.

Though it is not readily apparent, the Window Menu can actually be displayed from either a window or an icon. As we'll see, certain menu functions apply only to one or the other. This section describes using the Window Menu to perform various management functions on a window. (The sections "Pointer Commands to Manage Icons" and "Using the Windown Menu on Icons," later in this chapter, describe the use of Window Menu items, pointer commands, and other shortcuts on icons.)

The Window Menu command button is in the upper-left corner of the window frame and is identified by a short horizontal bar in its center. You can display the Window Menu from a window by moving the pointer to the command button and either:

- Clicking the first pointer button.
- Pressing and holding down the first pointer button.

If you've clicked the first pointer button to display the menu (the easier method), the first item that is available for selection is highlighted by a box. Figure 4-2 shows the default Window Menu, which has been displayed by clicking the first pointer button in the menu command button.

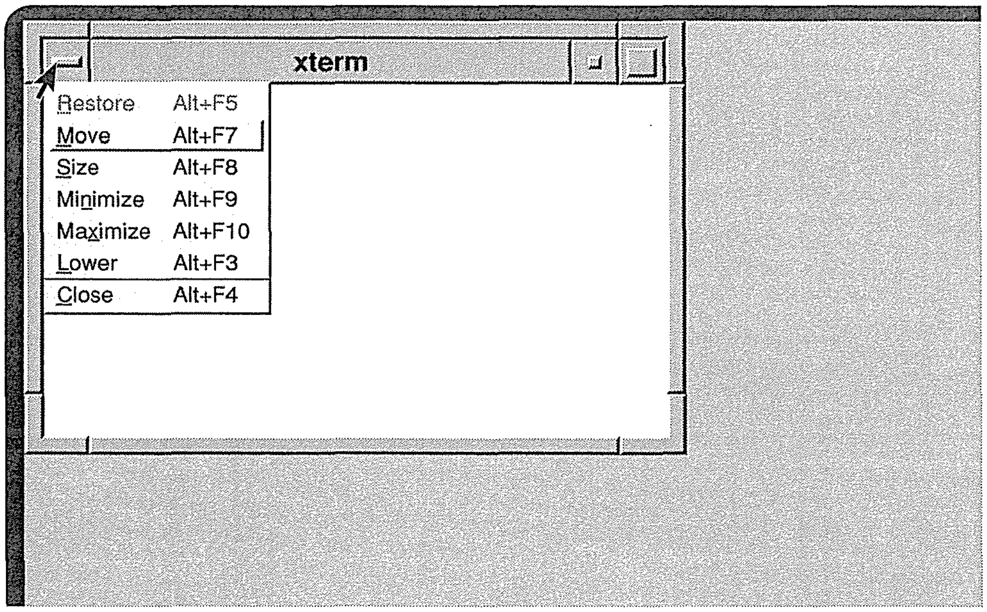


Figure 4-2. The Window Menu

You can also display the Window Menu by placing the pointer anywhere on the frame (other than the command buttons) and pressing and holding down the third button. The following keyboard shortcuts display the menu on the focus window: Shift-Escape or Meta-space.*

The function performed by each of the Window Menu items is fairly obvious. Six of the seven menu items (all but Lower) allow you to perform functions that can also be performed by simple pointer actions on the *mwm* window frame. Since effecting changes on a window using the frame is very simple and accessible, you will probably not use the Window Menu often.

You may want to use the menu to Lower a window (to the bottom of the stack), since this function cannot be performed by a simple pointer action on the frame. (If you learn the keyboard shortcut for this menu item, explained later in this section, you may not need the Window Menu to manage windows at all. You might also customize the *.mwmrc* file to include a key/button combination to lower a window. See Chapter 13 for details.) You may find the menu more helpful in managing icons, as described later in this chapter.

In any case, learning how to invoke the Window Menu items is helpful in orienting yourself within the Motif environment.

Note that if your keyboard does not have an F10 function key, the Maximize item will not appear on the Window Menu without 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. See Chapter 13, *Customizing mwm*, for information about the *.mwmrc* file.

Invoking Window Menu Items

Let's take another look at the Window Menu in Figure 4-2. Notice that the first item available for selection (indicated by the surrounding box) is Move. The first item on the menu, Restore, is used to change an icon back into a window or a maximized window back to its normal size; therefore, it is not useful at this time. The fact that Restore is not selectable is indicated by the fact that it appears in a lighter typeface.

Notice also that one letter of each menu item is underlined. This letter represents a unique abbreviation for the menu item, called a *mnemonic*, which can be used to select the item (when the menu is posted).

A keyboard shortcut follows each command. These shortcuts are known as *accelerators* because they facilitate the action. The keyboard accelerators allow you to perform all of the functions without having to display the menu (though they also work while the menu is displayed). To invoke an action using an accelerator, the window must have the input focus.

All of the keyboard accelerators for the menu items involve the Alt key and a function key. Remember that *mwm* considers the Alt and Meta keys to be equivalent. What this boils down to is that, for any of the shortcuts, you can substitute Meta for Alt.

*When a menu is displayed using a key or key combination, the same keystroke(s) can be used to remove it. Thus, Shift-Escape and Meta-space will also toggle the Window Menu off.

Once the Window Menu is displayed, you can select an item in the following ways:

- If you displayed the menu by pressing and holding down the first pointer button, drag the pointer down the menu to the desired item and release the first button.
- If you displayed the menu by clicking the first pointer button, either:
  - Move the pointer onto the item and click the first button.
  - Type the unique abbreviation or mnemonic (the underlined letter). (Though several of the abbreviations are capital letters, you should type the lowercase equivalent.)
  - Type the accelerator key combination. (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.

To remove the menu without making a selection, move the pointer off of the menu and release or click the first pointer button, as appropriate.

Most items work similarly to the comparable functions performed using the pointer on the frame. The primary difference relates to moving or resizing a window. Using the frame, you press and hold down a pointer button, move the pointer, and release the button to complete the action. Once you invoke the Move or Size item from the Window Menu (by any of the methods described previously), you simply move the pointer (without holding a button down); then click the first pointer button to complete the action.

If you test the various items, you'll find that each item works in a fairly predictable way. When you select Move, for instance, the pointer changes to the cross-arrow cursor, which appears in the center of the window; as you move the pointer, a window outline follows; you place the window in its new location by clicking the first pointer button. When you select Size, the pointer again changes to the cross-arrow cursor in the center of the window; move the pointer into any part of the resize border and the pointer symbol becomes one of the resize cursors; as you drag the border or corner, a window outline follows the pointer; then complete the resizing by clicking the first pointer button.

Pointer Commands to Manage Icons

In addition to managing windows, *mwm* provides several easy methods for managing icons. The following functions can be invoked using simple pointer button actions on an icon:

Move	Hold down the first pointer button and drag the icon to the desired position. Then release the button.
Raise	Click on the obscured icon with the first pointer button. The icon is raised to the top of the stack. (*mwm* does not allow icons to overlap one another; you'll need to raise an icon only when it's obscured by a window.) Note that this action also posts the Window Menu.

Restore (Deiconify)

To convert an icon back to a window, double click on the icon with the first pointer button. The window is displayed in the state it was in before it was iconified. (Thus, if the window was previously maximized, double clicking will convert the icon to the maximum size window.)

Each of these icon management function using the pointer is described in greater detail in the section "Managing Windows Using the mwm Frame" in Chapter 3, *Working in the X Environment.*

Using the Window Menu on Icons

You can also display the Window Menu from an icon and invoke menu items that affect it. To display the menu, just place the pointer on the icon and click the first button.* (You can also press and hold down the third button; or you can use either of these keyboard shortcuts: Shift-Escape or Meta-space.)

The Window Menu displayed from an icon is virtually identical to the menu displayed from a window; it contains all of the same items, but only five of the seven are selectable. (When displayed from a window, six of the seven items are selectable.) The five selectable items are: Restore, Move, Maximize, Lower, and Close. These items perform actions on an icon analogous to those performed on a window (see "Using the Window Menu" earlier in this chapter).

Two menu items, Size and Minimize, appear in a lighter typeface, indicating they are not available for selection. Size cannot be selected because, unlike a window, an icon cannot be resized. Obviously, Minimize cannot be used to iconify an icon.

Table 4-3 summarizes the Window Menu functions when invoked from an icon. For instructions on selecting an item and performing the various functions, read "Using the Window Menu" earlier in this chapter. Note that the keyboard shortcuts (accelerators) for the commands are also the same as those described for windows.

Table 4-3. Window Menu Actions on an Icon

Menu Item	Function	Shortcut
Restore	Converts the icon back to a window (in its previous state).	Alt+F5
Move	Moves the icon on the display.	Alt+F7
Size	Not available for selection.	n/a
Minimize	Not available for selection.	n/a

*This behavior is controlled by an *mwm* resource variable called `iconClick`, which is true by default. If the menu is not posted when you click the first button, check the setting for this variable. See Chapter 13, *Customizing mwm*, and the *mwm* reference page in Part Three of this guide. See Chapter 11, *Setting Resources*, for more information about resource file syntax.

Table 4-3. Window Menu Actions on an Icon (continued)

Menu Item	Function	Shortcut
Maximize	Converts an icon to a window the size of the root window.	Alt+F10
Lower	Sends an icon to the bottom of the window/icon stack.	Alt+F3
Close	Exits the client, removing the icon.	Alt+F4

To invoke a Window Menu action on an icon using the keyboard accelerator, the icon must have the input focus. As explained earlier in this chapter, you can direct focus to an icon by placing the pointer on it and clicking the first button (which also displays the Window Menu). The icon label becomes wider and the border is highlighted.

An icon also retains the input focus when you display the Window Menu and then remove it without selecting an item (by clicking anywhere outside the menu). The icon label will remain wide and the border highlighted until you direct focus to another window or icon. As long as an icon remains highlighted, you can invoke Window Menu commands using their keyboard accelerators.

In Chapter 13, we'll discuss using *mwm* resources to set up an icon box, a window for organizing icons on the display. Using an icon box changes the way you work with the Window Menu from an icon and introduces another menu item, Pack Icons, which reorganizes icons in the icon box. As we'll see in the next section, the 1.2 Root Menu offers the same function.

The Root Menu

The Root Menu is *mwm*'s main menu. Most of the commands it provides can be thought of as affecting the entire display. To display the Root Menu, move the pointer to the root window and press and hold down the third pointer button. The default Root Menu appears in Figure 4-3.

When you display the Root Menu, the pointer changes to the arrow pointer. As you can see, the default Root Menu offers only six items. To select an item, use the following steps:

1. As you continue to hold down the third pointer button, move the pointer onto the desired item name. (If you accidentally move the pointer off the menu, it will still remain displayed, as long as you continue to hold the third button down.) As you move the pointer onto an item, notice that a rectangular box is displayed around the item to highlight it.

2. Once the pointer is positioned on the item you want, release pointer button three. The action is performed.

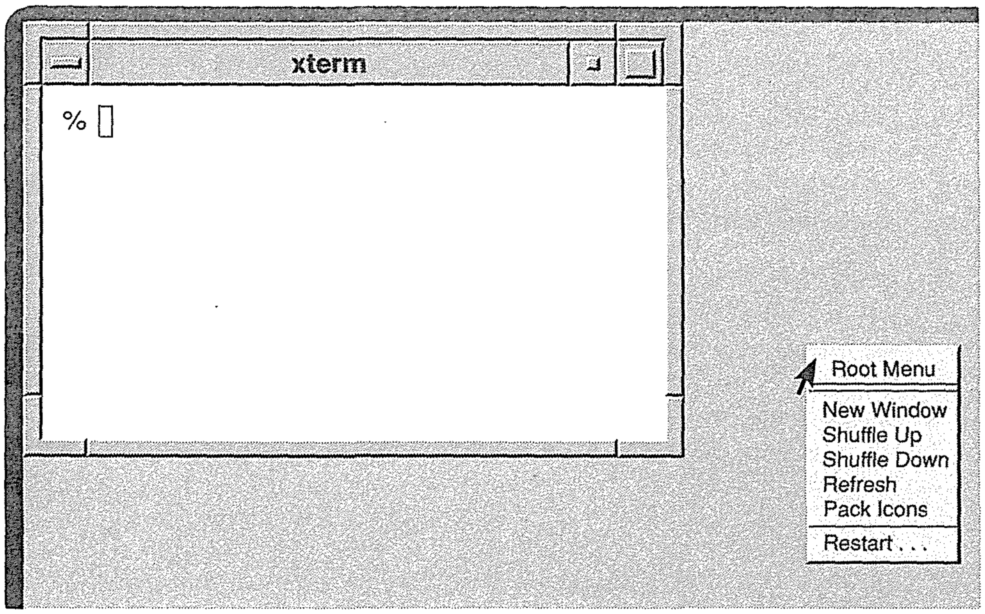


Figure 4-3. The mwm Root Menu

The functions performed by the default Root Menu are described below.

New Window	By default, this command runs an *xterm* window on the display specified by the DISPLAY environment variable, generally the local display. When you create a new window (by using the menu or typing the command in an *xterm*), the new window automatically becomes the active window.
Shuffle Up	If windows and/or icons are stacked on your display, this command moves the bottom window or icon in the stack to the top (raises it). (It's generally simpler to raise a window or icon by placing the pointer on it and clicking the first button, but this item is useful when an object is entirely obscured.)
Shuffle Down	If windows and/or icons are stacked on your display, this command moves the top window or icon in the stack to the bottom (lowers it).
Refresh	This command is used to *refresh* the display screen, that is, redraw its contents. Refresh is useful if system messages appear on the screen, overlaying its contents. (The *xrefresh* client can be used to perform the same function. Simply type `xrefresh` at the system prompt in an *xterm* window. If you own the "console," you can avoid many such problems by running the *xconsole* client, described in Appendix A, *Managing Your Environment*.)

Pack Icons	If you are using an icon box to organize icons on the display, this item optimizes the icons layout in the box. See Chapter 13, *Customizing mwm*, for instructions on setting up *mwm* to use an icon box. Note that the Window Menu provides the same item when displayed on the icon box window.
Restart...	Stops and restarts *mwm*. This is useful when you've edited the *.mwmrc* configuration file, which specifies certain *mwm* features, and want to activate the changes. Since this function is potentially more dangerous than the other Root Menu options, it is separated from the other options by a horizontal line.

When you select Restart, a dialog box appears in the center of the screen with command buttons asking you to either OK the restart process or Cancel the request. Click on the appropriate command button using the first pointer button. (In most cases, you should also be able to select the highlighted button simply by pressing Return or hitting a space.)

Note that the ellipse (...) following the menu item signals that you will be queried for confirmation. (Dialog boxes commonly appear in applications written using the Motif Toolkit. For more information, see "Dialog Boxes and Push Buttons" in Chapter 9, *Working with Motif Applications*.)

If you select OK, the window manager process is stopped. The screen will momentarily go blank. The new *mwm* process will be started immediately. While the new *mwm* process is starting, an hourglass symbol is displayed in the center of the otherwise blank screen. The hourglass appears to be filling up with sand until the window manager is running and the windows again are displayed on the screen.

Keep in mind that you can add, change, or remove menu items using the *mwm* configuration file, *.mwmrc*, in your home directory. We'll discuss customizing the Root Menu in Chapter 13.

5

The xterm Terminal Emulator

This chapter describes how to use xterm, *the terminal emulator. You use this client to create multiple terminal windows, each of which can run any programs available on the underlying operating system.*

In This Chapter:

5 The xterm Terminal Emulator

As we've seen, *xterm* provides you with a terminal within a window. Anything you can do using a standard terminal, you can do in an *xterm* window. Once you have an *xterm* window on your screen, you should be able to work productively immediately. From the *xterm* window, you can also run other clients.

To take advantage of X's windowing and networking capabilities, you'll probably want to run more than one *xterm* at a time (as well as other clients), perhaps on different systems in your network. (See Chapter 3, *Working in the X Environment*, for instructions on running clients on different systems using the `-display` option.)

Running multiple *xterm*s allows you to perform multiple tasks simultaneously—and to coordinate those tasks—neither of which you can do very successfully on a standard terminal. For instance, you can display the contents of a directory in one window while you edit a file in another window. Or you can have a program compiling in one window while you read mail in a second window. Note that, although you can display output simultaneously in several windows, you can type in only one window at a time.

Some important (and not so obvious) features of *xterm*: When you start an *xterm* process on the command line in one *xterm* window, the second *xterm* inherits the environment variables of the first (including the DISPLAY setting); the second shell also starts in the working directory of the first shell.

But *xterm* provides much more than basic terminal capabilities. Two of *xterm*'s most useful features are a scrollbar, which allows you to review text in the window, and a "copy and paste" facility, which allows you to select text from one window using the pointer and paste it into another (or even the same) window.

As we'll see, you can create an *xterm* window with a scrollbar using the `-sb` command-line option or specify a scrollbar as a default characteristic of *xterm* using the `scrollbar` resource variable. You can also add a scrollbar to (or remove one from) an *xterm* window already running on the display by using one of the client's four menus. Without customizing the client in any way, you can cut and paste text between *xterm* windows.

Among the less obvious features of *xterm* is a dual functionality. By default, *xterm* emulates a DEC VT102 terminal, a common alphanumeric terminal type. However, *xterm* can also emulate a Tektronix 4014 terminal, which is used to display graphics. For each *xterm* process, you can switch between these two types of terminal windows. You can display both a VT102 and a Tektronix window at the same time but only one of them can be the "active" window, i.e., the window receiving input and output. Hypothetically, you could be editing in the VT102 window while looking at graphics in the Tektronix window.

You switch between the VT102 window and the Tektronix window using items from certain *xterm* menus. *xterm* has four menus that can be used to control the VT102 and Tek windows, to select many terminal settings, and to run other commands that affect the *xterm* process.

Perhaps the most useful menu is the VT Fonts menu, you can change the font used to display text in the VT102 window. You may want to change the font for a number of reasons. Perhaps you need a larger font to read text more easily; or maybe you want to use a smaller font to reduce the size of a window while a program is running and you don't need to monitor its progress. Prior to this innovation (at Release 4), if you didn't like the display font, you had to start a new *xterm* process, specifying an alternative. The VT Fonts menu makes *xterm* much more flexible.

We'll take a look at some of the more useful items on each menu as well as some alternatives to menu items later in this chapter. For more complete information about menus, see the *xterm* reference page in Part Three of this guide.

We'll also consider how to run a program in a temporary *xterm* window, which goes away when the program finishes.

But first, let's consider some preliminary issues: which terminal type to specify for *xterm* and what to do when resizing an *xterm* window causes problems with its terminal emulation.

Then we'll look at the *xterm* features you'll probably use most frequently: the scrollbar and the text selection mechanism.

Terminal Emulation and the xterm Terminal Type

Anyone who has used a variety of terminals knows that they don't all work the same way. Each time you run *xterm*, it looks for a terminal type, which tells the system how the window should operate (i.e., the terminal type determines what sort of terminal *xterm* emulates). When *xterm* is assigned an inappropriate terminal type, the window does not always display properly, particularly when using a text editor such as *vi*. If your *xterm* windows seem to be displaying properly, chances are *xterm* is finding a valid terminal type and you can skip to the next section. If *xterm* is having emulation problems, read on.

When you're working with *xterm*, there are two ways the terminal type can be assigned. First, as in any UNIX environment, the terminal type can be specified in one of your shell startup scripts (e.g., *.login*, *.profile*, etc.). If you've recently switched from a standard UNIX environment to UNIX with X, one of your scripts probably does set a terminal type (maybe by setting the TERM environment variable, maybe by specifying multiple characteristics using *tset*(1), etc.). Second, if none of the startup files contains a terminal assignment, *xterm* automatically searches the database of terminal entries for the first suitable entry and sets the TERM environment variable accordingly.

xterm can emulate a variety of terminal types, which are listed on the client reference page in Part Three of this guide. An *xterm* window most successfully emulates a terminal when it has been assigned the terminal type "xterm."

If one of your shell startup scripts currently specifies a default terminal type, you (or your system administrator) will need to replace this with an appropriate type (preferably "xterm")—or remove the terminal type altogether. Deleting the terminal type assignment from all login files may actually be preferable. The only liability in deleting the terminal type line from a login file is that you might mistakenly delete other important settings. If necessary, consult your system administrator.

Note that for the "xterm" terminal type to be recognized on your system, the system administrator will have had to add it to the file containing valid *termcap* or *terminfo* entries. (The "xterm" entries are supplied with the standard release of X.) If this has not been done and you specify "xterm" in a startup file, *xterm* will assume a terminal type of "unknown" and you will have emulation problems. (This is another argument for deleting any terminal type assignment from a startup file.)

Regardless of whether you specify a terminal type or let *xterm* find its own entry, your system administrator should add "xterm" to the terminal database. See the *xterm* reference page in Part Three of this guide, and Volume Eight, *X Window System Administrator's Guide*, for more information. The Nutshell Handbook *termcap and terminfo* describes how to work with the terminal entry databases.

Resizing and Terminal Emulation

When you run *xterm*, the client sets the appropriate environment variables to reflect the dimensions of the window. Many programs use this information to determine the physical dimensions of output to the window.

If you resize an *xterm* window, the shell must be notified so that programs that rely on this information can work with the correct dimensions. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from BSD 4.3), *xterm* will use these facilities to notify programs running in the window whenever it is resized. However, if your operating system does not support terminal resizing capabilities, you may need to request explicitly that the environment variables be updated to reflect the resized window.

The *resize* client sends a special escape sequence to the *xterm* window and *xterm* sends back the current size of the window. The results of *resize* can be redirected to a file that can then be sourced to update TERMCAP (on *termcap* systems) or LINES and COLUMNS (on *terminfo* systems).

To update the appropriate variables to match a window's changed dimensions using the Bourne shell, enter:

```
$ resize > filename
```

and then execute the resulting shell command file:

```
$ . filename                    Bourne shell syntax
```

The variable(s) will be updated and the dimensions of the text within the window will be adjusted accordingly.

If your version of UNIX includes the C shell, you *source* the shell command file:

```
% source filename                    C shell syntax
```

However, in the C shell, it's preferable to define this alias for *resize*:

```
alias rs 'set noglob; eval `resize`; unset noglob'
```

Then use `rs` to update the variable(s) to reflect a window's new dimensions.

Note that even if your operating system supports terminal resizing capabilities, *xterm* may have trouble notifying programs running in the window that the window has been resized. On some older systems (based on BSD 4.2 or earlier), certain programs, notably the *vi* editor, cannot interpret this information. If you resize a window during a *vi* editing session, *vi* will not know the new size of the window. If you quit out of the editing session and start another one, the editor should know the new window size and operate properly. On newer systems (e.g., BSD 4.3 and later), these problems should not occur.

Running a Program in a Temporary xterm Window

Normally, when you start up an *xterm* window, it automatically runs another instance of the UNIX Bourne or C shell (depending on which is set in your *.Xresources* file or the SHELL environment variable). If you want to create an *xterm* window that runs some other program and goes away when that program terminates, you can do so with the *xterm* `-e` option:

```
% xterm -e command [arguments]
```

For example, if you want to look at the file *temp* in a window that will disappear when you quit out of the file, you can use the UNIX *more* program as follows:

```
% xterm -e more temp
```

When you are using other options to *xterm* on the command line, the `-e` option must appear last because everything after the `-e` option is read as a command.

Note that the titlebar of the *xterm* window will display the name of the command following `-e` (unless you've specified an alternative string using `-title`—see Chapter 10).

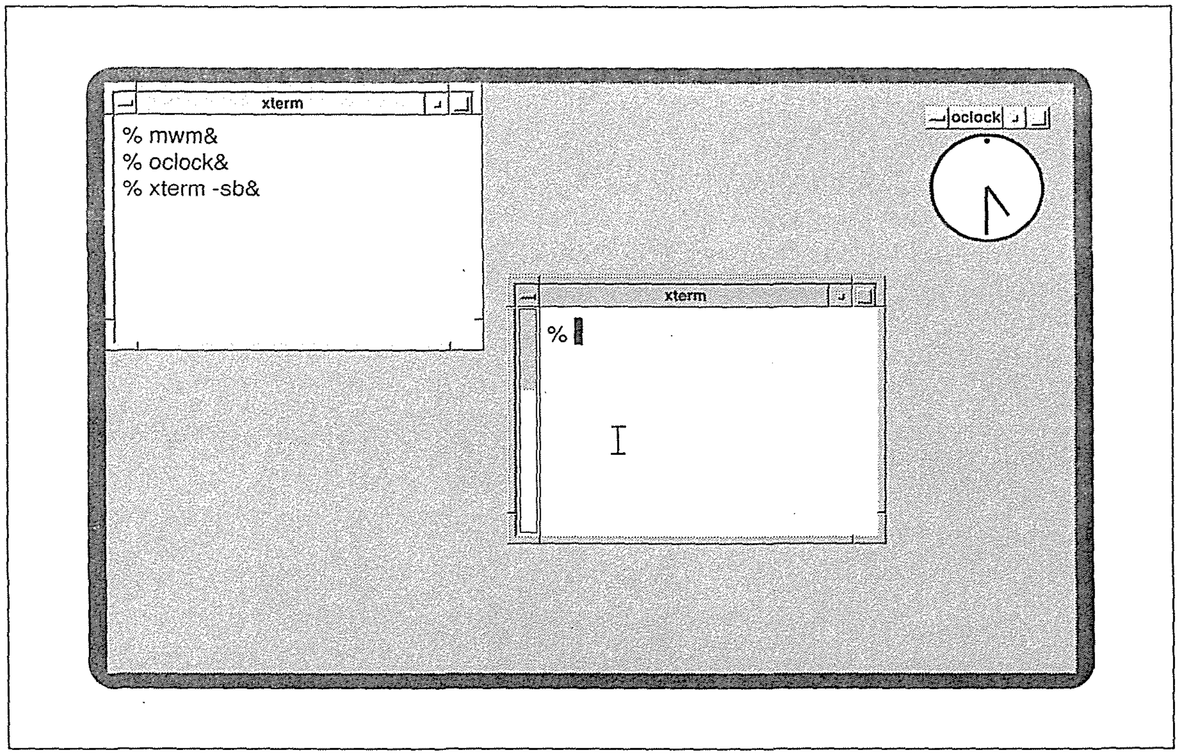
Running xterm with a Scrollbar

When using *xterm*, you are not limited to viewing the 24 lines displayed in the window at one time. By default, *xterm* actually remembers the last 64 lines that have appeared in the window. If the window has a scrollbar, you can scroll up and down through the saved text.

To create a single *xterm* window with a scrollbar, use the `-sb` command-line option:

```
% xterm -sb &
```

Figure 5-1 shows an *xterm* window with a scrollbar.



```
% mwm&
% oclock&
% xterm -sb&
```

```
% 
```

Figure 5-1. An xterm window with a scrollbar

To display all *xterm* windows with a scrollbar by default, set `scrollBar` in your *.Xresources* file, as described in Chapter 11. The appropriate resource setting is illustrated below:

```
XTerm*scrollBar: true
```

If an *xterm* window was not created with a scrollbar, you can add one using the Enable Scrollbar item on the VT Options menu. See the section "VT Options Menu" later in this chapter for instructions on selecting a menu item.

The Athena Scrollbar

Many applications provide horizontal and/or vertical scrollbars that allow you to look at a window's contents that extend beyond the viewing area. You move text (or images in graphics applications) in the window by placing the pointer on the scrollbar and performing some sort of action.

xterm's scrollbar is created by the Athena Scrollbar widget. (As we'll see in subsequent chapters, several of the standard X clients use Athena scrollbars.) An Athena scrollbar looks and operates differently than a scrollbar provided by a Motif application (that is, one created using the Motif widget set), as described in Chapter 9. If you're accustomed to using a Motif (or even a Macintosh) scrollbar, the Athena scrollbar may take some getting used to. While

Motif and Mac scrollbars have 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.

The Athena scrollbar has two parts: a *thumb* (the highlighted area within the scrollbar) which moves within the *scroll region*, as indicated in Figure 5-2.

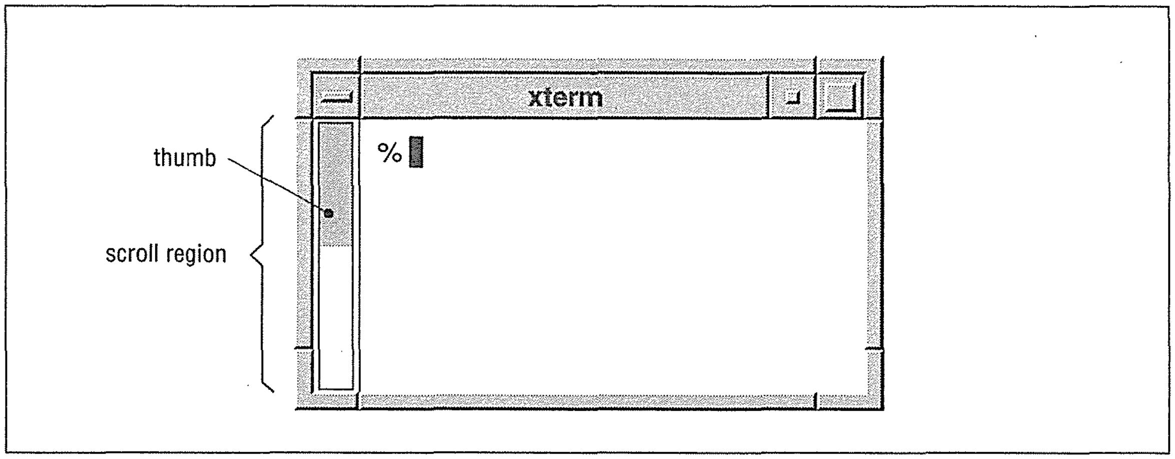


Figure 5-2. The parts of the Athena scrollbar

The thumb displays the position and amount of text currently showing in the window relative to the amount saved. When an *xterm* window with a scrollbar is first created, the thumb fills the entire scrollbar. As more text is saved, the size of the thumb decreases. The number of lines saved is 64 by default but an alternative can be specified with either the `-sl` command-line option or the `saveLines` value in an *.Xresources* file.

How to Use the Scrollbar

You scroll through the saved text using various pointer commands. When the pointer is positioned in the scrollbar, the cursor changes to a two-headed arrow. You can then scroll the text in various ways by clicking or dragging with certain pointer buttons.

Table 5-1 summarizes all of the scrollbar commands. However, the first command may be all you need to know. You can drag the text either up or down using the second pointer button. This command is the simplest and offers the most control over how much scrolling takes place. To drag the text in this manner:

1. Place the pointer on the scrollbar.
2. Press and hold down the second pointer button.
3. Then drag the thumb up and down.

Notice that text moves as you move the thumb. If you drag up, the window scrolls back toward the beginning of information saved in the window. If you drag down, the window scrolls forward toward the end of information in the window. When you release the button, the window displays the text at that location. This makes it easy to get to the top of the data by pressing the second button, dragging the thumb to the top of the scroll region, and releasing the pointer button.

To get back to the current cursor position, press any key. Either the space bar or Return is a good choice.

Note that there are three additional scrollbar commands, listed in Table 5-1 and explained subsequently. If you're satisfied to drag the scrollbar using the second pointer button, feel free to skip ahead and learn something about copying and pasting text.

Table 5-1. Athena Scrollbar Commands

To move text in this direction:	Place pointer on scrollbar and:	Notes:
Either up or down	Hold down second pointer button and drag thumb.	Text follows pointer movement.
Down	Click first pointer button.	Scrolls towards latest saved text (towards bottom of window).
Up	Click third pointer button.	Scrolls towards earliest saved text (towards top of window).
Either up or down	Click second pointer button.	Scrolls to a position in saved text that corresponds to the pointer's position in scroll region.

The next three pointer commands in Table 5-1 involve a click that causes the text to scroll However, if you test them, you'll find that it's difficult to judge how much text you're going to scroll with a single click.

Clicking the first pointer button in the scrollbar causes the window to scroll toward the end of information in the window.

Clicking the third pointer button in the scrollbar causes the window to scroll toward the beginning of information in the window.

Clicking the second pointer button moves the display to a position in the saved text that corresponds to the pointer's position in the scroll region. For example, if you move the pointer to the very top of the scroll region and click the second button, the window scrolls to a position very near the beginning of the saved text. As you might imagine, it's very difficult to guess how much scrolling will take place when you use the scrollbar in this way.

Copying and Pasting Text Selections

Once your *xterm* window is created, you can select text to copy and paste within the same or other *xterm* windows using the pointer. You don't need to be in a text editor to copy and paste. You can also copy or paste text to and from the command line.

Text copied into memory using the pointer is saved in a global cut buffer and also becomes what is known as the PRIMARY text "selection."* Both the contents of the cut buffer and the contents of the PRIMARY text selection are globally available to all clients. When you paste text into an *xterm* window, by default the contents of the PRIMARY selection are pasted. If no text is in the PRIMARY selection, the contents of the cut buffer (called CUT_BUFFER0), are pasted. In most cases, these will be the same and you don't have to think about it. As we'll see later, this background to the text selection mechanism becomes important when you want to perform certain customizations, particularly those involving the *xclipboard* client.

For now, however, let's just consider the standard methods for copying and pasting text between *xterm* windows.

Selecting Text to Copy

There are several ways to select (copy) text, all using the pointer. You can select a passage of text, or you can select text by individual words or lines. Table 5-2 summarizes all of the text selection methods. Hypothetically, the passage can be of any length. However, the size of the window limits the amount of text you can copy at one time; also keep in mind that there can be problems pasting long selections.

In order to copy text from a window, the window must have the input focus. The click to focus input is not interpreted as an attempt to start a text selection.

There are two methods for selecting a passage of text. The simpler way is to:

1. Click the first pointer button at the beginning of the text you want to select.
2. Move the pointer to the end of the text selection and click the third pointer button.

The text between the marks is highlighted and copied into memory. (Technically speaking, the text is copied into CUT_BUFFER0—a global cut buffer—and is also made the PRIMARY text selection.)

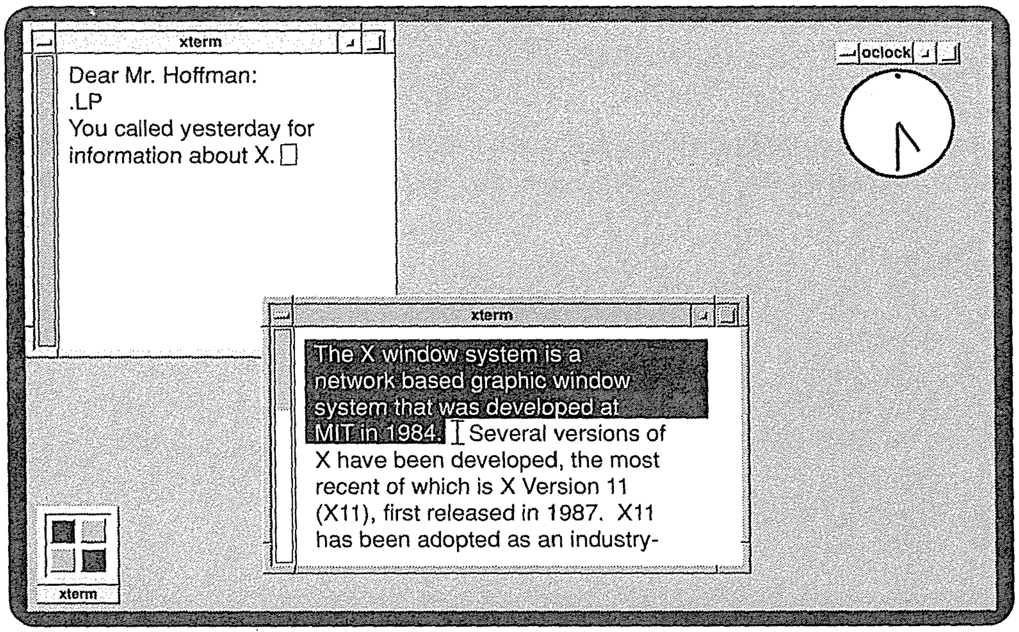
As an alternative, you can make the selection by dragging the pointer:

1. Place the pointer at the beginning of the text you want to select.
2. Hold down the first button.

*The PRIMARY selection and the cut buffer are stored as *properties* of the root window. A property is a piece of information associated with a window (or font) and stored in the server, where it can be accessed by any client. The property mechanism permits "cut" text to be stored and later "pasted" into the windows of other clients. See Chapter 1 and Chapter 11 for more about properties and interclient communication.

3. Drag the pointer to the end of the desired text.
4. Release the button.

The text is highlighted and copied into memory (i.e., copied into the global cut buffer and also made the PRIMARY selection, as in Figure 5-3).



Dear Mr. Hoffman:
.LP
You called yesterday for
information about X.

The X window system is a
network based graphic window
system that was developed at
MIT in 1984. Several versions of
X have been developed, the most
recent of which is X Version 11
(X11), first released in 1987. X11
has been adopted as an industry-

Figure 5-3. Highlighted text saved as the PRIMARY selection

You can select a single word or line simply by clicking. To select a single word, place the pointer on the word and double-click the first button.* To select a single line, place the pointer on the line and triple-click the first button.

Table 5-2 lists the possible pointer actions and the selections they make. You always begin by placing the pointer on the text you want to select.

*To be more precise, double-clicking selects all characters of the same class (e.g., alphanumeric characters). By default, punctuation characters and whitespace are in a different class from letters or digits—hence, the observed behavior. However, character classes can be changed. For example, if you wanted to double-click to select email addresses, you'd want to include the punctuation characters !, %, @, and . in the same class as letters and digits. However, redefining the character classes is not something you'd do every day. See the *xterm* reference page in Part Three of this guide for details.

Table 5-2. Button Combinations to Select Text for Copying

To select	Do this
Passage	Click the first button at the start of the selection and the third button at the end of the selection. Or: At the beginning of the selection, hold down the first button; move the pointer to the end of the desired text; and release the button.
Word	Double-click the first button anywhere on the word.
Line	Triple-click the first button anywhere on the line.

Each selection replaces the previous contents of CUT_BUFFER0 and the previous PRIMARY text selection. You can make only one selection at a time. (The *xclipboard* client, described later in this chapter, can be used to store multiple text selections.)

To clear the highlighting, move the pointer off the selection and click the first button anywhere else in the window. Note, however, that the text still remains in memory until you make another selection.*

Extending a Selection

Regardless of how you make a selection, you can extend that selection, generally using the third pointer button. You can extend a selection in a few ways. To learn what is perhaps the simplest way, follow these steps:

1. Bring up *vi* (or any other text editor with which you are familiar) in an *xterm* window, and type in this sample sentence:

```
The X Window System is a network-based graphics window system that
was developed at MIT in 1984.
```

2. Place the pointer at the beginning of the word *network-based* and click the first button.

3. Then move the pointer to the end of *graphics* and click the third button. The words *network-based graphics* are selected:

```
The X Window System is a network-based graphics window system that
was developed at MIT in 1984.
```

4. Then move the pointer away from the selected words to the left or right to encompass additional text. For the purposes of our example, let's move the pointer to the beginning of the sentence. Then click the third pointer button. A new selection now extends from

*Technically speaking, this action clears the PRIMARY selection, but the text remains in CUT_BUFFER0. When you subsequently try to paste the text, *xterm* finds the PRIMARY selection empty, so (according to its defaults) it pastes the contents of the cut buffer instead. See Chapter 11, *Setting Resources*, for more information about these so-called *xterm translations*.

the previous selection (*network-based graphics*) to the pointer's location and looks something like this:

```
The X Window System is a network-based graphics window system that
was developed at MIT in 1984.
```

Alternatively, you can press and hold down the third pointer button and drag the pointer to extend the selection. Then release the pointer button.

Remember that an extension always begins from the previous selection. By moving the pointer up or down, or to the right or left of the last selection, you can select part of one line or add or subtract several lines of text.

If the previous selection was by word or line (by double- or triple-clicking), when you extend it, the extension is automatically by word(s) or line(s). There are a few ways of extending a selection made by word or line.

First, if you hold the button down after double- or triple-clicking (rather than releasing it) and move the pointer, you will select additional text by words or lines at a time. Then release the button to end the selection.

More commonly you will probably decide to extend the selection after making it (and releasing the first pointer button). Under these circumstances, you can extend it as in the following example:

1. Starting again with our sample sentence, place the pointer on the word *graphics* and select it by double-clicking the first pointer button.

   ```
   The X Window System is a network-based graphics window system that
   was developed at MIT in 1984.
   ```

2. Then move the pointer to the right, onto any part of the word *window*, and click the third button. The new selection encompasses *graphics* and the entire word *window*.

   ```
   The X Window System is a network-based graphics window system that
   was developed at MIT in 1984.
   ```

3. You can also click to extend the selection by multiple words. Try moving the pointer to the left this time, to the middle of the word *System*, and click the third button. The new selection encompasses every word between *System* and *window*, inclusive:

   ```
   The X Window System is a network-based graphics window system that
   was developed at MIT in 1984.
   ```

If you originally selected an entire line by triple-clicking the first pointer button, moving the pointer to another line and clicking the third button extends the selection to encompass that new line and all lines in between.

As an alternative, you can also extend a word or line selection by pressing and holding down the third pointer button, dragging the pointer, and releasing the third button. The extension still increments by word or line as appropriate.

Table 5-3. Button Combinations to Extend a Text Selection

To select	Do this
By passage	Move the pointer to the place in the text to which you want the selection to extend; click the third button. Or: Hold down the third button; move the pointer to the end of the text you want to include; and release the button.
By word	Move the pointer to the word to which you want the selection to extend (either to the left or right of the previous selection); click the third button anywhere on the word. Or: Hold down the third button; drag the pointer onto the last word you want to include (either to the left or right of the previous selection); release the button. Or: After double-clicking the first button anywhere on the word to select it, continue to hold the button down and drag the pointer to the left or right. The selection will be extended by word. When you've included the words you want, release the button.
By line	Move the pointer to the line to which you want the selection to extend (either above or below the previously selected line); click the third button anywhere on the line. Or: Hold down the third button; drag the pointer onto the last line you want to include (either above or below the previously selected line); release the button. Or: After triple-clicking the first button anywhere on the line to select it, continue to hold the button down and drag the pointer up or down. The selection will be extended by line. When you've included the lines you want, release the button.

To select text that fills more than one screen, select the first screenful. Use the scrollbar to view the additional text. Then use the third pointer button to extend the selection. The original selection does not need to be in view; clicking the third button will extend it to the point you choose.

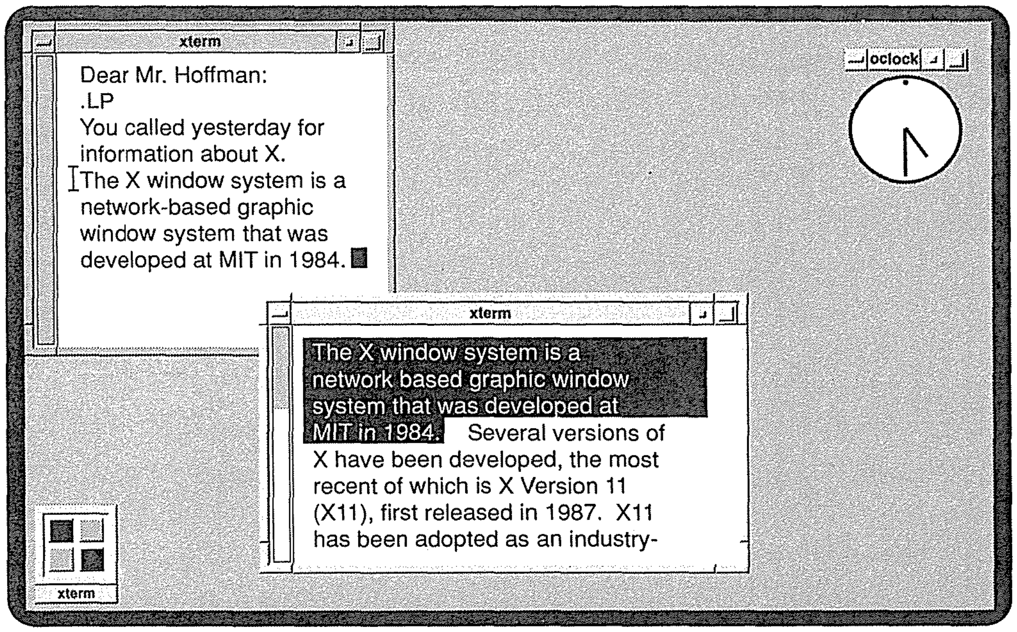
To clear the highlighting, move the pointer off the selection and click the first button anywhere else in the window. Note, however, that the text still remains in memory until you make another selection.

Complications can arise if you're copying text that includes tabs. With the current implementation of the copy and paste feature, tabs are saved as spaces. If you're copying a large amount of text with many tabs from one text file to another, having tabs converted to spaces can create problems. A possible workaround is to change all tabs in the first file to some unique character or string (using a global command provided by your text editor); copy and paste the text into the second file; convert the unique strings back to tabs in both files using your text editor.

Pasting Text Selections

Clicking the second button inserts the text from the PRIMARY selection (or CUT_BUFFER0, if the selection is empty) as if it were keyboard input. You can move data from one *xterm* window to another by selecting the data in one window with the first button (or extending the selection with the third button), moving the pointer to another window, and clicking the second button.

You can paste text either into an open file or at a command-line prompt. To paste text into an open file, as illustrated in Figure 5-4, click the second button within the window containing the file. The text from the memory area will be inserted at the text editor cursor. (Of course, the file must be in a mode where it is expecting text input, such as the insert mode of an editor.) You can paste the same text as often as you like. The contents of the PRIMARY selection remain until you make another selection.



Dear Mr. Hoffman:
.LP
You called yesterday for
information about X.
The X window system is a
network-based graphic
window system that was
developed at MIT in 1984.

The X window system is a
network based graphic window
system that was developed at
MIT in 1984. Several versions of
X have been developed, the most
recent of which is X Version 11
(X11), first released in 1987. X11
has been adopted as an industry-

Figure 5-4. Pasting text into an open file

To paste text at a command-line prompt, you must first close any open file within the window. Then click the second button anywhere within the window to place the text on the command line at the end of text in the window. (Note that the window will scroll to the bottom on input.) You can make multiple insertions by repeatedly clicking the second button.

Note that you can paste text into a window when click-to-type focus is in effect, even if the window does not have the input focus. The act of pasting does not transfer focus either.

(Similarly, if you click on a window to focus input, the click is not interpreted as an attempt to start a text selection.)

Keep in mind that you can paste *over* existing text in a file with the *vi* change text commands (such as cw, for change word). For example, you can paste over five words by specifying the *vi* command 5cw, and then pasting text by clicking the second pointer button. Note that you can paste over existing text in any editor that has an overwrite mode.

The Text Selection Mechanism and xclipboard

Prior to Release 3, many clients exchanged information solely by means of global cut buffers, which are, in effect, owned by the server, and available to all clients. Cut buffers are useful only for copying and pasting information that does not need to be translated to another format, such as ASCII text between two *xterm* windows.

In accordance with the newer interclient communication conventions developed since Release 2, most clients, notably *xterm*, primarily exchange information via selections. The advantage of the selection mechanism is that it allows data from one client to be converted to a different format to be used by another client. Cut buffers do not perform this type of translation.

A selection is globally available but not owned by the server. A selection is owned by a client—initially by the client from which you copy it. Then when the text selection is pasted in another window, that window becomes the owner of the selection.

The selection mechanism has a couple of limitations you should be aware of, although it's likely only one of them will present a problem. Because of the rules of precedence governing cut buffers and selections and the nature of selections (particularly the issue of ownership), the following problems can arise in transferring data:

1. By default, you can save only one selection at a time.
2. For a selection to be transferred to a client, the selection must be owned by a client. If the client that owns the selection no longer exists, the transfer cannot be made.

The *xclipboard* client can address both of these problems.

In the next section, we'll show you how to use *xclipboard*. To work properly with *xclipboard*, you need to do some customization, which is explained in Chapter 11, *Setting Resources*. The short story is that the *xclipboard* window is a storehouse for text that is copied to what is known as the CLIPBOARD selection. Like the PRIMARY selection, the CLIPBOARD selection is another *property* of the root window—a piece of information stored in the server where it is available to any client.

Most users will probably not encounter the second problem. You are probably doing all of your copying and pasting between *xterm* windows. If you've made a selection from an *xterm* window and the window is killed, the *selection* contents are lost. However, the cut buffer contents remain intact and are pasted instead. (Since all *xterm* windows interpret ASCII text, the translation capabilities of the selection mechanism are not needed.)

Problems involving the loss of selections are more likely to happen if you are transferring information between clients that require information to be in different formats. Although *xclipboard* is primarily intended to allow you to store multiple text selections, it can also avert problems of selection ownership by providing centralized ownership (via the CLIPBOARD selection). Once the CLIPBOARD owns a selection, the selection can be transferred (and translated to another format), even if the client that previously owned the selection goes away.

You can customize a client to send data to the CLIPBOARD selection by using *event translations*, which are discussed in Chapter 11. The client you'll probably want to use in conjunction with *xclipboard* is *xterm*; Chapter 11 suggests appropriate translations to do this. To use another client with *xclipboard*, see the client reference page in Part Three of this guide for information on the appropriate translations. For more information on selections and translations, see Volume One, *Xlib Programming Manual*.

Saving Multiple Selections with xclipboard

The *xclipboard* client provides a window in which you can paste multiple text selections and from which you can copy text selections to other windows. Similar to the clipboard feature of the Macintosh operating system, the *xclipboard* is basically a storehouse for text you may want to paste into other windows, perhaps multiple times. The *xclipboard* window is shown in Figure 5-5.

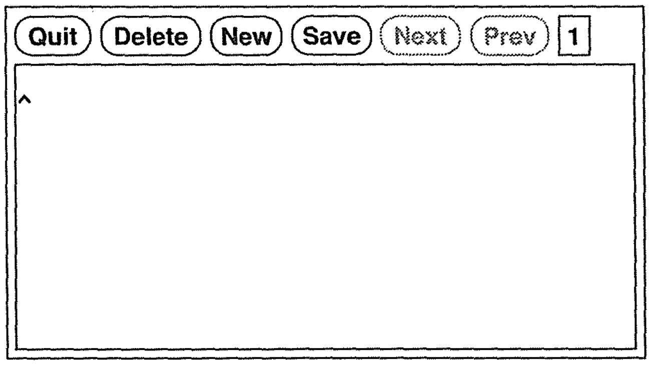


Figure 5-5. The xclipboard window

To open an *xclipboard*, type:

```
% xclipboard &
```

You can paste text into the *xclipboard* window using the pointer in the manner described previously and then copy and paste it elsewhere but this is not its intended use. To use the *xclipboard* most effectively, you must do some customization involving a resource file, such as *.Xresources*. The necessary steps are described in detail in Chapter 11. For now, suffice it to say that you want to set up the *xclipboard* so that you can select text to be made the CLIPBOARD selection and have that text *automatically pasted* in the *xclipboard* window, as illustrated in Figure 5-5.

Since the *xclipboard* client is intended to be coordinated with the CLIPBOARD selection, the X server allows you to run only one *xclipboard* at a time.

In order to illustrate how the clipboard works, let's presume it has been set up according to the guidelines in Chapter 11. According to those guidelines, you make text the CLIPBOARD selection by:

1. Holding down the first pointer button and dragging the pointer to highlight the text (one of the usual selection methods); and then,

2. while continuing to hold the first button, clicking the third button. (Then you can release button 1.)

(You could specify another button combination or a button and key combination but we've found this one works pretty well.) The first pointer action makes the text the PRIMARY selection (and it is available to be pasted in another window using the pointer); the second pointer action additionally makes the text the CLIPBOARD selection (and it is automatically sent to the *xclipboard* window, as in Figure 5-6).

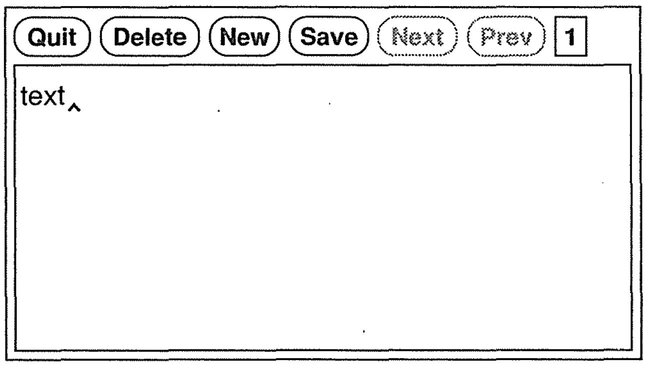


Figure 5-6. Selected text appears automatically in the xclipboard window

These guidelines still allow you to select text with the first pointer button alone and that text will be made the PRIMARY selection; however, the text will not automatically be sent to the *xclipboard*. This enables you to make many selections but to direct to the *xclipboard* only those selections you consider important (perhaps those you might want to paste several times). (The guidelines in Chapter 11 give you different ways to paste the PRIMARY and CLIPBOARD selections, as we'll see later in this section.)

In order to allow you to store multiple text selections, the seemingly tiny *xclipboard* actually provides multiple screens, each of which can be thought of as a separate buffer. (However, as we'll see, a single text selection can span more than one screen.) Each time you use the pointer to make text the CLIPBOARD selection, the *xclipboard* advances to a new screen in which it displays and stores the text. Several command buttons allow you to manage the *xclipboard* window and its selections.

To the right of the command buttons is a tiny box which displays a number corresponding to the current CLIPBOARD selection. This handy box has been added in Release 5. When the client is first run, the box displays the number "1;" the number is automatically incremented for each additional selection (or reduced by 1 if you select the Delete command button).

Once you have saved multiple selections, the client's Next and Previous command buttons allow you to move forward and backward among these screens of text. (The box to the right of the command buttons displays the number of the selection currently in the window to help you keep track.)

Pasting the CLIPBOARD selection has different ramifications than pasting the PRIMARY selection. When you paste the PRIMARY selection (by clicking the second pointer button), you always get the last PRIMARY selection you made (by any of several acceptable methods—see "Selecting Text to Copy" earlier in this chapter).

When you paste the CLIPBOARD selection, you get the selection that's currently being displayed in the *xclipboard* window (which is not necessarily the last one you sent there). For example, you might send four selections to the CLIPBOARD, then use the Prev button to go back to selection #2. If you then paste the CLIPBOARD selection, selection #2 is pasted.

If you've coordinated *xterm* with *xclipboard* using the guidelines outlined in Chapter 11, you paste the CLIPBOARD selection in an *xterm* window by:

1. Holding down the Shift key.
2. Clicking the second pointer button.

Remember that you generally paste a text selection by clicking the second pointer button; we've added the Shift key to distinguish the CLIPBOARD selection from the PRIMARY selection.

The functionality of the client's command buttons is summarized in Table 5-4. They are all selected by clicking the first pointer button.

Table 5-4. xclipboard Command Buttons and Functions

Button	Function
Quit	Causes the application to exit.
Delete	Deletes the current *xclipboard* buffer; the current screenful of text is cleared from the window and the next screenful (or previous, if there is no next) is displayed.

Table 5-4. xclipboard Command Buttons and Functions (continued)

Button	Function
New	Opens a new buffer into which you can insert text; the window is cleared.
Save	Allows you to save the currently displayed text selection to a file. Pops up a dialog box with a text window displaying a default filename (*clipboard*) and two command buttons: Accept and Cancel. Clicking on Accept or pressing Return saves the current selection as the file *clipboard* (in the directory from which you ran the program). Or you can change the filename using the same commands used with *xedit* (see Chapter 8) before hitting Accept. You can only Save one selection at a time (subsequent saves overwrite the filename); so if you want to save multiple selections, you'll need to change the filename each time. To bail out without saving, click on Cancel.
Next and Previous	Once you have sent multiple selections to the *xclipboard*, Next and Previous allow you to move from one to another (e.g., display them sequentially). Before two or more CLIPBOARD selections are made, these buttons are not available for use. (Their labels will appear in a lighter typeface to indicate this.)

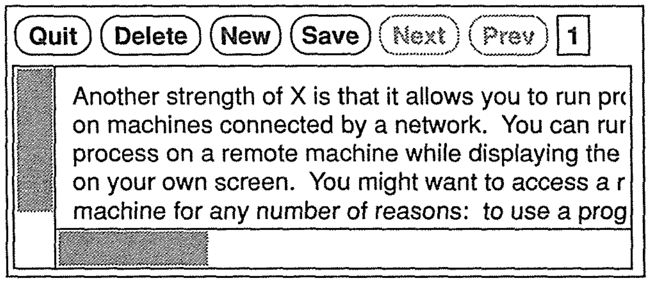
The command buttons you will probably use most frequently are Delete, Next, and Previous.

When you select text using the first and third pointer buttons, the text will automatically be displayed in the *xclipboard* window and will, in effect, be the first screenful of text (or first buffer) saved in the *xclipboard*. Subsequent CLIPBOARD selections will be displayed and saved in subsequent screens.

You can remove a screenful of text from the *xclipboard* by displaying that screenful and then clicking on the Delete command button. When you delete a screenful of text using this command button, the next screenful (if any) will be displayed in the window. If there is no next screenful, the previous screenful will be displayed.

Certain features of *xclipboard* become apparent only when you make a very large CLIPBOARD selection. Say you select a full *xterm* window of text with the first and third pointer buttons, as described above. The text extends both horizontally and vertically beyond the bounds of a single *xclipboard* screen. (As we suggested earlier, a CLIPBOARD selection can actually span more than one *xclipboard* screen. Pressing Delete will remove all screensful the selection comprises.) When you make a selection that extends beyond the bounds of the *xclipboard* screen (either horizontally, vertically, or both), scrollbars will be activated in the window to allow you to view the entire selection.

If the text extends both horizontally and vertically beyond the bounds of the *xclipboard* screen, as it does in Figure 5-7, the window will display both horizontal and vertical scrollbars. If the text extends beyond the screen in only one of these two ways, the window will display either a horizontal or vertical scrollbar, as needed.* These scrollbars are selection-specific: they are only displayed as long as the current selection cannot be viewed in its entirety without them. If you move to a previous or subsequent selection that *can* be viewed without scrollbars, the scrollbars will be deactivated.



Another strength of X is that it allows you to run prc
on machines connected by a network. You can rur
process on a remote machine while displaying the
on your own screen. You might want to access a r
machine for any number of reasons: to use a prog

Figure 5-7. xclipboard with scrollbars to view large text selection

Problems with Large Selections

If you experiment making large selections with *xclipboard*, you may discover what seems to be a bug in the program. Though in most circumstances, making a new selection causes the screen to advance and display the new text, this does not happen reliably after a selection vertically spanning more than one screenful. In these cases, the new selection *is* saved in the *xclipboard* (and the number in the small box is incremented to indicated this); however, the *xclipboard* window does not automatically advance to show you the new current selection. Instead, the previous long selection is still displayed. (For example, though the box says "5," indicating that a fifth selection has been saved, the window is still displaying selection #4.) This is a bit of *xclipboard* sleight-of-hand. The new selection has been successfully made but the appearance of the window belies this fact. (The Next button will probably add to your confusion; it will not be available for selection, suggesting that the text in the window is the last selection saved. This is not the case.)

In order to get around this problem and display the actual current selection, press the Previous button. The same long selection (which is, in actuality, the Previous selection) will again be displayed. (The small box will flip back to display the preceding number as well.) Then the Next button will be enabled, and you can click on it to display the actual current selection. (The selection displayed in the window and the number in the small box will correspond.)

*An application created using the X Toolkit, which provides horizontal and vertical scrollbars, is described as a *viewport*. See Chapter 8 for more information about viewports and other X Toolkit features.

Editing Text Saved in the xclipboard

You can edit text you send to the *xclipboard* using the same commands recognized by *xedit*. These commands are described in the section "The xedit Text Editor" in Chapter 8. A small caret cursor will be visible in each screenful of text. You can move this cursor by clicking the pointer where you'd like it to appear. Then you can backspace to delete letters or type to insert them. When you edit a screenful of text, the *xclipboard* continues to store the edited version, until you delete it or exit the program.

Be aware that, without performing customization, you can still use *xclipboard* on a very simple level. You can paste text into and copy text from the *xclipboard* window just as you would any other, using the pointer movements described earlier in this chapter. You can also type in the *xclipboard* window and then copy and paste what you've typed. Just move the pointer into the window and try typing. However, keep in mind that this is not the intended use of the *xclipboard*.

If you do choose to use the clipboard in a limited way, it can still be a helpful editing tool. For example, say you wanted to create a paragraph composed of a few lines of text from each of two files. You could copy the text from each file using the pointer and paste it into the *xclipboard* window. (Each time you paste text into the *xclipboard* window, the text is appended to whatever text was already pasted there.) Again using the pointer, you could copy the newly formed paragraph from the *xclipboard* window and paste it into a file in another window.

The xterm Menus

xterm has four different menus, each providing items that serve different purposes. You display a menu by placing the pointer on the window and simultaneously pressing the Control (keyboard) key and a pointer button. (The exact key and button combinations are described in subsequent sections with each menu.) When you're using a window manager, such as *mwm*, that provides a titlebar or frame, the pointer must rest within the window proper—not on any window decoration. (Note that the pointer must be within the window, even if click-to-type focus is enabled. See Chapter 1 for a discussion of focus policy.)

The following menus are available:

- Main Options menu
- VT Options menu
- VT Fonts menu
- Tek Options menu

As shown in Figure 5-8, three of the four *xterm* menus are divided into sections separated by horizontal lines. The top portion of each divided menu contains various modes that can be toggled. (The one exception is the Redraw Window item on the Main Options menu, which is a command.) A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state.

The items on the VT Fonts menu change the font in which text is displayed in the *xterm* window. Only one of these fonts can be active at a time. To toggle one off, you must activate another.

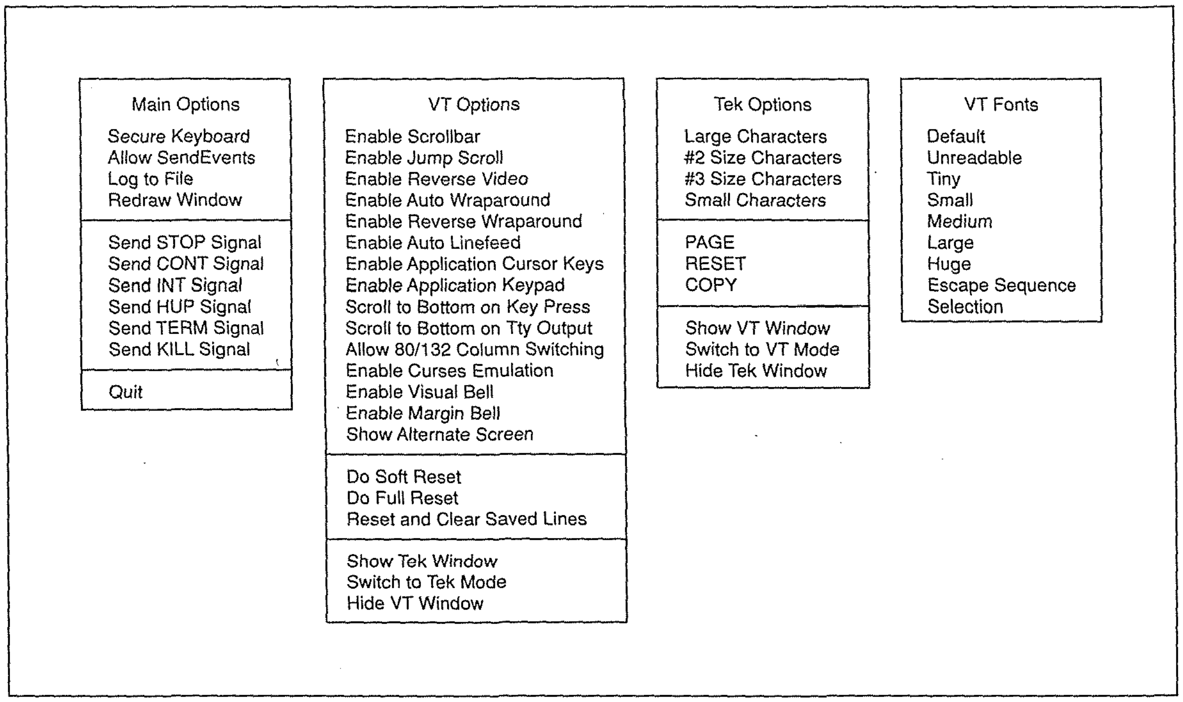


Figure 5-8. The Release 5 xterm menus

Most mode entries can also be set by command-line options when invoking *xterm*, or by entries in a resource startup file (such as *.Xdefaults* or *.Xresources*) as described in Chapter 11. (See the *xterm* reference page in Part Three of this guide for a complete list of command options and resource variables.) The various modes on the menus are very helpful if you've set (or failed to set) a particular mode on the command line and then decide you want the opposite characteristic.

The sections below the modes portion of each menu contain various commands. Selecting one of these commands performs the indicated function. Many of these functions can only be invoked from the *xterm* menus. However, some functions can be invoked in other ways: for example, from an *mwm* menu, on the command line, by a sequence of keystrokes (such as Control-C). This chapter includes alternatives to some of the menu items which, in certain cases, may be more convenient. Of course, the *xterm* menus can be very helpful when other methods to invoke a function fail.

When you display an *xterm* menu, the pointer becomes the arrow pointer and initially appears in the menu's title. Once the menu appears, you can release any keyboard key. The menu will remain visible as long as you continue to hold down the appropriate pointer button. (You can move the pointer off the menu without it disappearing.)

If you decide not to select a menu item after the menu has appeared, move the pointer off the menu and release the button. The menu disappears and no action is taken.

We think users will be most interested in the mode toggles on the VT Options menu (that allow you to turn features like the scrollbar on and off) and the items on the VT Fonts menu (that allow you to change the display font once the client is running). For the short story on *xterm* menus, just take a look at these sections.

The remaining sections of this chapter survey additional menu items and also discuss some alternatives to using the menus. For a brief description of all menu items, see the *xterm* reference page in Part Three.

The Main Options Menu

The Main Options menu, shown in Figure 5-9, allows you to set certain modes and to send signals (such as SIGHUP) that affect the *xterm* process.

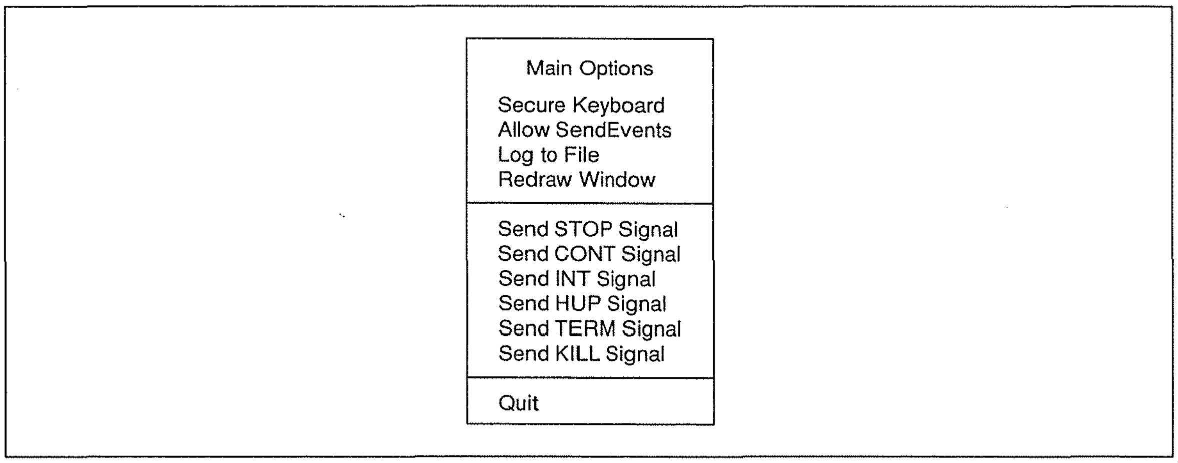


Figure 5-9. The Main Options menu

To bring up the Main Options menu, move the pointer to the *xterm* window you want to change, hold down the Control key, and press the first (usually the left) pointer button.* The pointer changes to the menu pointer, and this menu of three modes and eight commands appears. (You can release the Control key but must continue to press the first pointer button to hold the Main Options menu in the window.) Note that Main Options menu items apply only to the *xterm* window the pointer is in when you display the menu. To effect changes in another *xterm*, you must move the pointer to that window, display the menu, and specify the items you want.

*The right button can be made to function as the "first" button. This is especially useful if you are left-handed. See Chapter 14, *Setup Clients*, for instructions on how to customize the pointer with *xmodmap*.

To select a menu item, move the menu pointer to that item and release the first button. After you have selected a mode (Secure Keyboard, Allow SendEvents, or Log to File), a check mark appears before the item to remind you that it is active. The Log to File mode on the Main Options menu can also be set by a command-line option when invoking *xterm*. In addition, both Log to File and Allow SendEvents can be set by entries in a resource startup file such as *.Xresources*. The menu selections enable you to change your mind once *xterm* is running. (See the *xterm* reference page in Part Three for more information on these modes.)

The Secure Keyboard mode toggle is intended to help counteract one of the security weaknesses of X. You may want to activate it before you type a password or other important text in an *xterm* window. Generally, when you press a keyboard key or move the pointer, the X server generates a packet of information that is available for other clients to interpret. These packets of information are known as *events*. Moving the pointer or pressing a keyboard key causes input events to occur.

There is an inherent security problem in the client-server model. Because events such as the keys you type in an *xterm* window are made available via the server to other clients, hypothetically an adept system hacker could access this information. (Naturally, this is not an issue in every environment.) A fairly serious breach of security could easily occur, for instance, if someone were able to find out a user's password or the *root* password. Enabling Secure Keyboard mode causes all user input to be directed *only* to the *xterm* window itself.

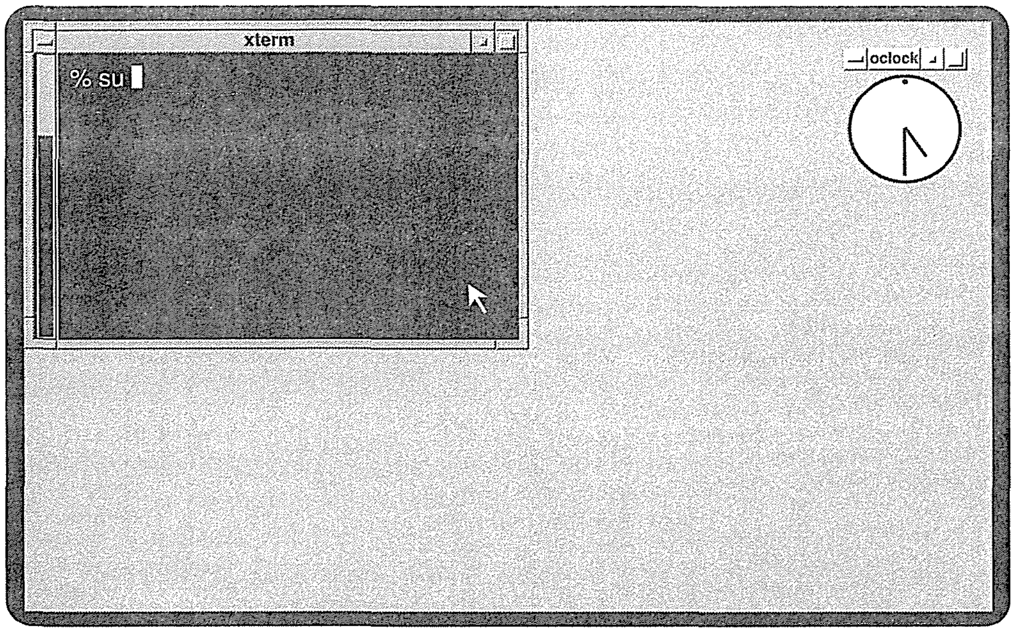
Of course, in many environments, precaution is probably not necessary: if the nature of the work is in no way sensitive, if the system administrator has taken pains to secure the system in other ways, etc. If your environment might be vulnerable, you can enable Secure Keyboard mode before typing passwords and other important information and then disable it again using the menu.

When you enable Secure Keyboard mode, the foreground and background colors of the *xterm* window will be exchanged (as if you had enabled the Reverse Video mode from the VT Options menu), as shown in Figure 5-10. When you disable Secure Keyboard mode, the colors will be switched back.

Be aware that only one X client at a time can secure the keyboard. Thus, if you have enabled Secure Keyboard mode in one *xterm*, you will not be allowed to enable it in another *xterm* until you disable it in the first. If Secure Keyboard mode is not available when you request it, the colors will not be switched and a bell will sound.

If you request Secure Keyboard mode and are not refused but the colors are *not* exchanged, be careful: you are not in Secure Keyboard mode. If this happens, there's a good chance that someone has tampered with the system. If the application you're running displays a prompt before asking for a password, it's a good idea to enable Secure Keyboard mode before the prompt is displayed and then verify that the prompt is displayed in the proper colors. Before entering the password, you can also display the Main Options menu again and verify that a check mark appears next to Secure Keyboard mode.

Be aware that Secure Keyboard will be disabled automatically if you iconify the *xterm* window, or start *mwm* or another window manager that provides a titlebar or other window decoration. (You can enable Secure Keyboard mode once *mwm* is running, though.) This limitation is due to the X protocol. When the mode is disabled, the colors will be switched back and the bell will sound to warn you.



```
% su
```

Figure 5-10. Reverse video is enabled when the keyboard is secure

Though intended to counteract a security weakness, the Secure Keyboard mode toggle can also be used to get around a weakness of *mwm*. As described in Chapter 4, if *mwm* dies, it's possible that the focus can be lost—i.e., the focus is no longer directed to any application window. Selecting Secure Keyboard mode for any *xterm* should cause that window to grab the focus again.

In addition to modes that can be toggled, the Main Options menu includes several commands. All of the commands (except for Redraw Window) send a signal that is intended to affect the *xterm* process: suspend it (Send STOP Signal), terminate it (Send TERM Signal), etc. Given that your operating system may recognize only certain signals, every menu item may not produce the intended function.

Note that most of these commands are equivalent to common keystroke commands, which are generally simpler to invoke. For example, in most terminal setups Control-C can be used to interrupt a process. This is generally simpler than using the Send INT Signal menu command, which performs the same function.

Similarly, if your system supports job control, you can probably suspend a process by typing Control-Z and start the process again by typing Control-Y, rather than using the Send STOP Signal and Send CONT Signal menu commands. If your system does not support job control, neither the menu commands nor the keystrokes will work.

Four of the commands (Send HUP Signal, Send TERM Signal, Send KILL Signal, and Quit) send signals that are intended to terminate the *xterm* window. Depending on the signals your system recognizes, these commands may or may not work as intended. Be aware that in most cases you can probably end an *xterm* process simply by typing some sequence (such as Control-D or `exit`) in the window. Of course the menu items may be very helpful if the more conventional ways of killing the window fail. Also be aware that, in addition to being recognized only by certain systems, some signals are more gentle to systems than others. See the *xterm* reference page in Part Three of this guide for information on the signal sent by each of the menu commands and the *signal*(3C) reference page in the *UNIX Programmer's Manual* for more information on what each signal does.

The Quit command sends a SIGHUP to the process group of the process running under *xterm*, usually the shell. (The Send HUP Signal command sends the same signal.) This ends up killing the *xterm* process, and the window disappears from the screen.

Quit is separated from the earlier commands by a horizontal line so it's easier to point at. Sending a SIGHUP with Quit is slightly more gentle to the system than sending a SIGKILL with Send KILL Signal.

The Redraw Window command redraws the contents of the window. As an alternative, you can redraw the entire screen using the *xrefresh* client. See the *xrefresh* reference page in Part Three of this guide for more information about this client.

VT Options Menu

The VT Options menu provides many VT102 setup functions, which can be used to specify certain characteristics of the *xterm* window. Some of these mode settings are analogous to those available in a real VT102's setup mode; others, such as *scrollbar*, are *xterm*-only modes.

The VT Options menu items allow you to reset several characteristics/modes at once, select the Tektronix window to accept input, hide the VT window, etc.

To bring up the VT Options menu, move the pointer to the *xterm* window, hold down the Control key, and then press and hold down the second pointer button. (You can release the Control key but must continue to press the second button to keep the VT Options menu in the window.) The menu shown in Figure 5-11 appears.

Check marks indicate the active modes. For example, Jump Scroll, Auto Wraparound, and Scroll to Bottom on Tty Output* are active in the VT Options menu displayed in Figure 5-11.

These are the only modes active by default. To turn off one of these modes, move the menu pointer to that mode and release the second button.

*This mode indicates that if you are using the scrollbar and the window receives output (or a key is pressed, if `stty echo` is enabled), the window scrolls forward so that the cursor is at the current line. (You can use the menu to toggle off this mode but it is generally desirable to have.)

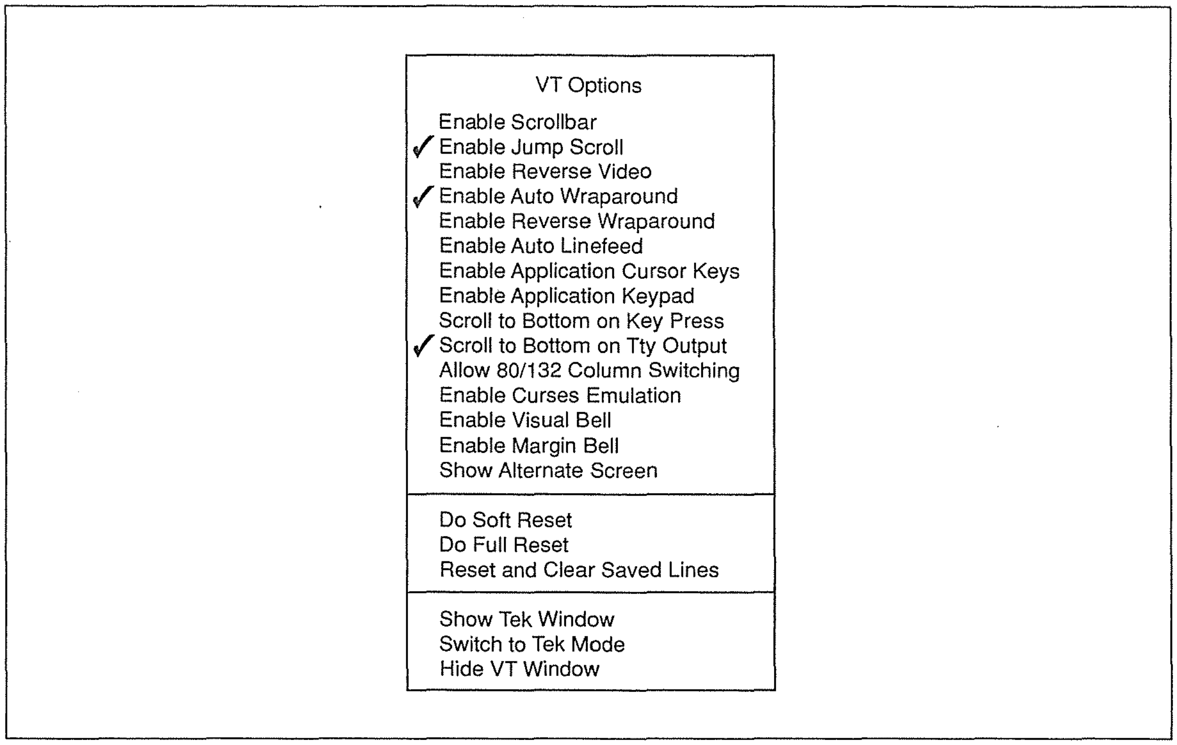


Figure 5-11. The VT Options menu

Most of these modes can also be set by command-line options when invoking *xterm* or by entries in a resource startup file like *.Xresources* (see Chapter 11). The menu selections allow you to change your mind once *xterm* is running.

The toggle Allow 80/132 Column Switching warrants a little more explanation. This mode allows *xterm* to recognize the DECCOLM escape sequence, which switches the terminal between 80- and 132-column mode. The DECCOLM escape sequence can be included in a program (such as a spreadsheet) to allow the program to display in 132-column format. See Appendix E, *xterm Control Sequences*, for more information. This mode is off by default.

The VT Options menu commands (in the second and third partitions of the menu) perform two sets of functions, neither of which can be performed from the command line or a resource definition file. The commands Soft Reset and Full Reset reset some of the modes on the menu to their initial states. See the *xterm* reference page in Part Three of this guide for more information.

The Show Tek Window, Switch to Tek Mode, and Hide VT Window menu items allow you to manipulate the Tektronix and VT102 windows.

The Show Tek Window command displays the Tek window and its contents without making it the active window (you can't input to it). Use the Switch to Tek Mode command to display a Tektronix window and make it the active window. When you select Switch to Tek Mode, the Show Tek Window command is automatically enabled, since the Tek window is displayed. (Note that a Tektronix window is not commonly used for general purpose terminal emulation but for displaying the output of graphics or typesetting programs.)

Both of these commands are toggles. If Show Tek Window is active and you toggle it off, the Tek window becomes hidden. (As we'll see, you can also do this with the Hide Tek Window item on the Tek Options menu.) If both Switch to Tek Mode and Show Tek Window are active (remember, enabling the former automatically enables the latter), toggling off either one of them switches the *xterm* back to VT mode. (This can also be done from the Tek Options menu with the Switch to VT Mode item.)

The Hide VT Window command hides the VT102 window but does not destroy it or its contents. It can be restored (and made the active window) by choosing Select VT Mode from the Tek Options menu.

VT Fonts Menu

The VT Fonts menu allows you to change the display font of an *xterm* window while the window is running, a powerful and very useful capability. To bring up the VT Fonts menu, move the pointer inside the *xterm* window. Press and hold down the Control key on the keyboard and press the third (usually the right) pointer button. The VT Fonts menu* is shown in Figure 5-12.

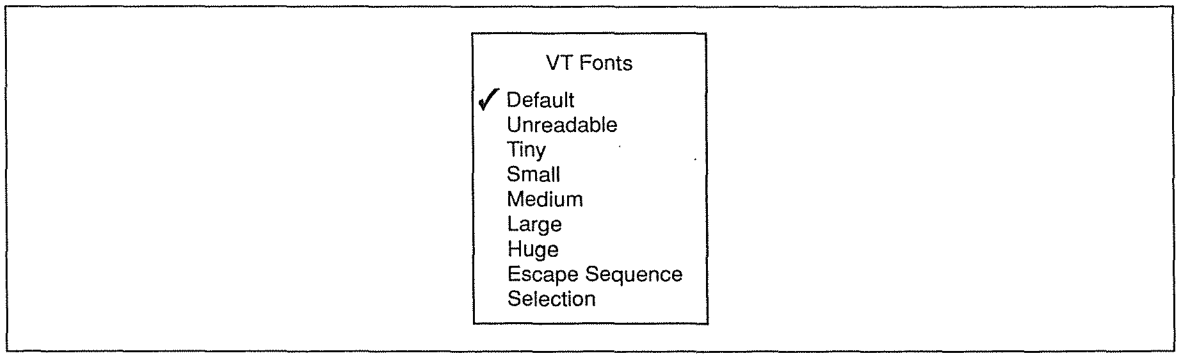


Figure 5-12. VT Fonts menu

If you have not toggled any items on this menu, a check mark will appear before the Default mode setting. The Default is the font specified when the *xterm* window was run. This font could have been specified on the *xterm* command line or in a resource file such as *.Xresources*. Whatever the case, this font remains the Default for the duration of the current *xterm* process.

The items Default, Unreadable, Tiny, Small, Medium, Large, and Huge can be toggled to set the font displayed in the *xterm* window. The font can be changed any number of times to accommodate a variety of uses. You might choose to use a large font for editing a file (chances are you've chosen a large enough default font, though). You could then change to a

*This menu has changed slightly in Release 5. The Huge item is entirely new. The Unreadable label is new in R5, but it toggles the same font the Tiny label did in R4. (If you've selected it, you know it has been aptly renamed.) The R5 Tiny item toggles a newly included, somewhat more legible choice.

smaller font while a process is running since you don't need to be reading or typing in that *xterm*. Changing the font also changes the size of the window.

There are also default settings for the Unreadable, Tiny, Small, Medium, Large, and Huge fonts. They are all constant-width fonts from the directory */usr/lib/X11/fonts/misc* and are listed in Table 5-5.

Table 5-5. VT Fonts Menu Defaults

Menu Item	Default Font
Unreadable	nil2
Tiny	5x7
Small	6x10
Medium	7x13
Large	9x15
Huge	10x20

Bring up the VT Fonts menu and toggle some of these fonts to see what they look like. If you select the Unreadable font (*nil2*), your *xterm* window becomes very tiny, almost the size of some application icons. Though you cannot read the actual text in a window this size, the window is still active and you *can* observe if additional output, albeit minuscule, is displayed. An *xterm* window displaying text in such a small font can, in effect, serve as an *active icon*.

Be aware that you can specify your own Unreadable, Tiny, Small, Medium, Large, and Huge fonts using entries in a resource startup file such as *.Xresources*. The corresponding resource names are `font1`, `font2`, `font3`, `font4`, `font5`, and `font6`. See Chapter 6 for more information about available fonts. See Chapter 11 for instructions on how to set resource variables.

In addition to the menu selections we've discussed, the VT Fonts menu offers two other possible selections: Escape Sequence and Selection. When you first run an *xterm* window, these selections appear on the VT Fonts menu but they are not functional. (They will appear in a lighter typeface than the other selections, indicating that they are not available.) In order to enable these selections for use, you must perform certain actions which are outlined in Chapter 6.

Tek Options Menu

The Tek Options menu controls certain modes and functions of the Tektronix window. The menu can only be displayed from within the Tektronix window. As previously described, you can display the Tek window and make it the active window by using the Switch to Tek Mode command on the VT Options menu.

To display the Tek Options menu, move the pointer inside the Tektronix window. Press and hold down the Control key on the keyboard and press the second pointer button. The Tek

Options menu appears. With this menu you set the size of the text in the Tektronix window and select some commands.

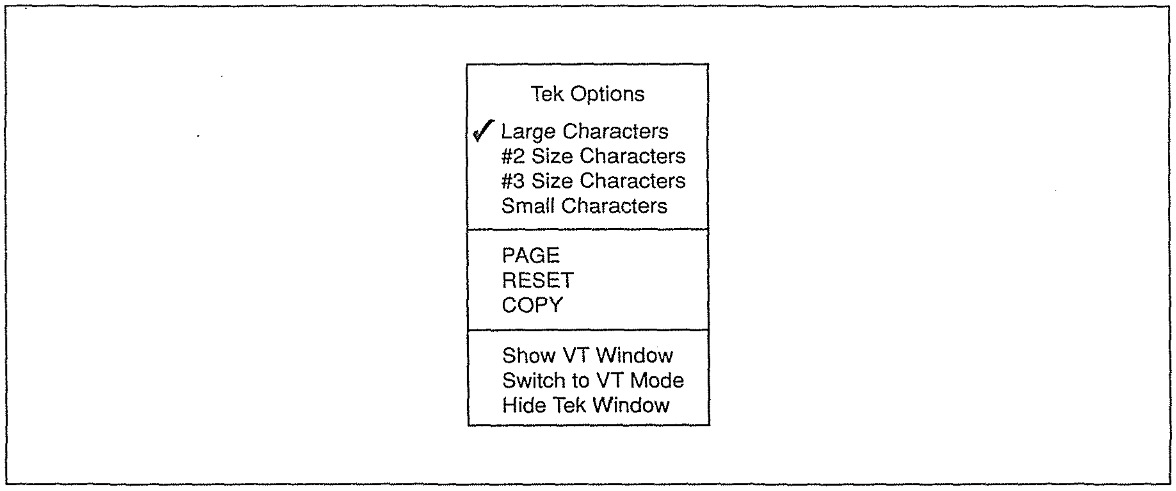


Figure 5-13. The Tek Options menu

Note that these modes (above the first line) can only be set from the Tek Options menu. All of these modes set the point size of the text displayed in the Tektronix window. (Only one of these four modes can be enabled at any time.)

The most important command on the Tek Options menu, shown in Figure 5-13, is Switch to VT Mode. If the Tek window has been made the active window (using the Switch to Tek Mode command from the VT Options menu), you can choose Switch to VT Mode to make the VT window the active window again. (If both windows are showing, you can also toggle Switch to Tek Mode on the VT Options menu to *deactivate* it; that is, switch *from* Tek mode and back to VT mode.) Switch to VT Mode is also a toggle; if you deactivate it, *xterm* will switch back to Tek mode.

Selecting Show VT Window displays the VT window if it has been hidden (using the Hide VT Window command from the VT Options menu) or hides it if it is being displayed. (Again, the command is a toggle.) Remember that you cannot input to the VT window until you make it the active window by using Switch to VT Mode.

6

Font Specification

This chapter describes what you need to know in order to select display fonts for the various client applications. After acquainting you with some of the basic characteristics of a font, this chapter describes the rather complex font naming conventions and how to simplify font specification. This chapter also describes how to use the xlsfonts, xfd, *and* xfontsel *clients to list, display, and select available screen fonts. Finally, this chapter explains the basics of using the font server (a Release 5 innovation) to access fonts resident on other machines on the network.*

In This Chapter:

6
Font Specification

Many clients allow you to specify the font used to display text in the window, in menus and labels, or in any other text fields. For example, you can choose the font used for the text in *mwm* menus or in *xterm* windows.

Unfortunately, for the most part, there are no simple "font menus" like there are on systems such as the Macintosh.* Instead, X has a fairly complex font naming system (which, like most things about X, is designed for maximum flexibility rather than for simplicity or ease of use). Of course, there will no doubt soon be many applications, such as word processors and publishing packages, that provide a simple interface for selecting fonts. However, for the clients in the X distribution, you are generally limited to selecting fonts via command-line options or resource specifications.

This wouldn't be so bad if a typical font name weren't mind-bending at first glance. Imagine typing this command line to create an *xterm* window whose text is to be displayed in a 20-point constant width font:

```
% xterm -fn -misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1
```

Fortunately, you can use asterisks as wildcards to simplify this name to a somewhat more reasonable one:

```
% xterm -fn '*fixed-medium-r-*-200-*'
```

and you can define even simpler aliases, so that you could end up typing a command line similar to this:

```
% xterm -fn 20p
```

In this chapter, we're going to try to make sense out of the sometimes perplexing and arcane information about fonts under X. First, we'll explain the font naming convention in detail. Along the way, we'll acquaint you with the appearance of some of the basic font families (groups of related fonts), and the various permutations (such as weight, slant, and point size) within each family.

Then, we'll talk about how to use font name wildcards to simplify font specification. We'll also talk about the font search path (the directories where the font files are stored), and how to define aliases for font names.

*An exception is *xterm*'s VT Fonts menu. But even in this case, you need to know a lot about font naming to change the fonts available on the menu.

X also provides several utilities for dealing with fonts on the local machine:

- *xlsfonts*, which lists the names of the fonts available on your server, as well as any aliases.
- *xfd* (font displayer), which allows you to display the character set for any individual font you specify on the command line.
- *xfontsel* (font selector), which allows you to preview fonts and select the name of the one you want. (This name can then be pasted onto a command line, into a resource file, etc.)

Once we have an understanding of font naming conventions, we'll consider how to use these clients.

In addition, we'll see how to use two options available on *xterm*'s VT Fonts menu.

Note that the current chapter also covers two Release 5 additions, scalable fonts and a font server, which are introduced in the following section.

If you don't want to venture too far into the somewhat bewildering jungle of font names and directories, skip to the section "If You Just Want to Pick a Font," which should cut through the underbrush.

What's New in Release 5?

Prior to Release 5, all of the standard fonts provided with X were *bitmap fonts*. Bitmap fonts require a separate font file for each size of the font. There are several liabilities to this arrangement, not the least of which is that having multiple files for basically the same typeface makes managing fonts more difficult and wastes disk space. Also, because of the different resolution of computer monitors, a font intended to be a particular point size might actually appear larger or smaller on various screens.

Release 5 sees the addition of *font scaling* and includes some new *outline fonts*, which are suitable for scaling. An outline font stored in a single file can be scaled to any point size you request (though the scaling requires some system overhead). Although bitmap fonts *can* be scaled, the resulting scaled font is usually jagged and somewhat illegible. The *Speedo* font directory in the standard distribution contains Charter and Courier outline fonts donated by Bitstream, Inc. For more information about the standard font directories and outline fonts, see "The Font Search Path" and "Specifying Scalable Fonts," respectively.

The standard Release 5 bitmap fonts should be available on your system in *portable compiled font* format. The font files have an extension of *.pcf* to indicate this format. The currently available outline fonts in the *Speedo* directory deviate somewhat from the norm—they have the extension *.spd*.

Also prior to Release 5, fonts needed to be available on the local machine or had to be provided over the network via certain protocols. In Release 5, the X Window System provides a *font server* (*fs*) from which you can request fonts resident on other machines in the network. Later in this chapter, we'll show you how to run the font server with a very simple configuration and how to add the font server to your font search path.

Release 5 also includes a few clients intended to be run when the font server is in use, among them:

- *fslsfonts*, which is analogous to *xlsfonts*, lists the names of the fonts available on the specified font server.
- *fsinfo* gives information about the font server, including the alternate servers that are available.
- *showfont** lists some of the more esoteric characteristics of a font and provides ASCII representations of the individual characters. You can use the output of *showfont* to convert a font character into a bitmap image using the *atobm* client, described in Chapter 7, *Graphics Utilities*.

After considering the font server, we'll take a closer look at these clients.

If You Just Want to Pick a Font

X font names can be so complicated that you might prefer not to deal with them at all. If you want to experiment with fonts or access fonts on remote machines, you'll have to spend some time familiarizing yourself with the conventions anyway. But if you just want to locate some fonts to use with *xterm* and other clients, you can use the predefined aliases for some of the constant width fonts that should be available on most systems. (Later in this chapter we'll discuss how to define your own aliases.)

Figure 6-1 lists the aliases for some constant width fonts that should be appropriate for most of the standard clients, including *xterm*. To give you an idea of the range of sizes, each alias is written in the font it identifies.

In these cases, the aliases refer to the dimensions in pixels of each character in the font. (For example, *10x20* is the alias for a font with characters 10 pixels wide by 20 pixels high.) Note, however, that an alias can be virtually any character string.

The default font for many applications, including *xterm*, is a 6 × 13 pixel font that has *two* aliases: "fixed" and "6x13." Many users consider this font to be too small. If you have enough screen space, you might want to use the 10 × 20 font for *xterm* windows:

```
% xterm -fn 10x20 &
```

You can make this font the default for *xterm* by specifying it as the value for the `font` resource variable (in an *.Xresources* or other resource file):

```
XTerm*font:  10x20
```

See Chapter 11, *Setting Resources*, for instructions on specifying and loading resources.

*Don't confuse the MIT client *showfont* with our public domain client *xshowfonts*, which was used to create the pictures in Appendix B, *Release 5 Standard Fonts*.

See "Font Name Aliasing" later in this chapter for instructions on surveying the predefined aliases (and creating aliases of your own).

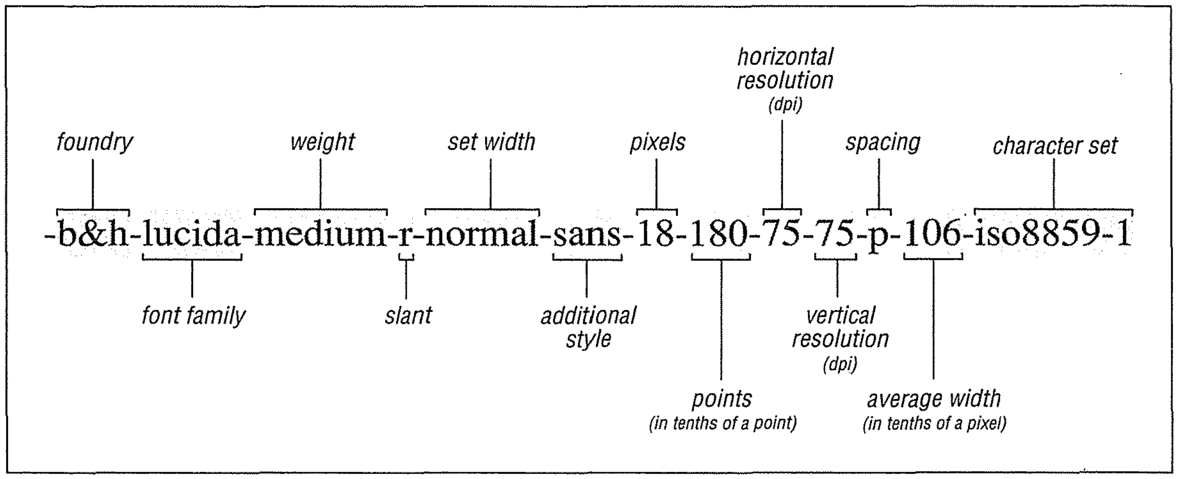
The section "Changing Fonts in xterm Windows" (later in this chapter) describes how to use two of the items on the *xterm* VT Fonts menu. Keep in mind, however, that you may need to know a bit more about font naming conventions, wildcards and aliasing to take full advantage of these options.

Font Naming Conventions

The X Window System font naming conventions are intended to allow for complete specification of all of the characteristics of each font. Unfortunately, this completeness makes the font names somewhat difficult to work with—at least until you learn what all the parts of the names mean, and get a handle on which parts you need to remember and which you can safely ignore. (By the end of this chapter, you should have that knowledge.)

The *xlsfonts* client can be used to display the names of all the fonts available locally to your server. When you run *xlsfonts*, you'll get an intimidating list of names similar to the name in Figure 6-1. Upon close examination, this rather verbose name contains a great deal of useful information: the font's developer, or foundry (b&h, Bigelow & Holmes); the font family (Lucida); weight (medium); slant (roman); set width (normal); additional style (sans serif); size of the font in pixels (18); size of the font in tenths of a point (180 tenths of a point, thus 18 points); horizontal resolution (75 dpi); vertical resolution (75 dpi); spacing (p, for proportional); average width (106—measured in tenths of a pixel, thus 10.6 pixels); and character set (iso8859-1).

As mentioned earlier, font name wildcarding can eliminate lots of unnecessary detail. If you are already familiar with font characteristics, skip ahead to the section "Font Name Wildcarding," later in this chapter, for some tips and tricks. If you need a refresher on fonts, read on as we illustrate and explain each of the elements that make up the font name.



horizontal resolution (dpi)
foundry
weight
set width
pixels
spacing
character set
-b&h-lucida-medium-r-normal-sans-18-180-75-75-p-106-iso8859-1
font family
slant
additional style
vertical resolution (dpi)
points (in tenths of a point)
average width (in tenths of a pixel)

Figure 6-1. Font name components

Font Families

It has been several years since the advent of desktop publishing and, by now, it is unlikely that anyone in the computer industry is unaware that text can be displayed on the screen and printed on the page using different fonts.

However, the term *font* is used somewhat ambiguously. Does it refer to a family of typefaces (such as Times® Roman or Helvetica®), which comes in different sizes, weights, and orientations? Or should each distinct set of character glyphs be considered a separate font?

For the most part, X takes the latter approach. When the documentation says that Release 5 includes more than 500 fonts, this sounds either intimidating or impressive, depending on your mood. But, in fact, the R5 X distribution includes only eight font families (Charter, Courier, Helvetica, Lucida®, New Century Schoolbook®, Symbol, Times, and the Clean family of fixed-width fonts), plus several miscellaneous fonts that are found only in individual sizes and orientations,* and many more special purpose fonts. (R5 also includes a few outline fonts that can be scaled to any size you specify. These represent exceptions—different sizes are not considered different fonts. For further discussion, see "The Size of Bitmap and Outline Fonts" later in this chapter.)

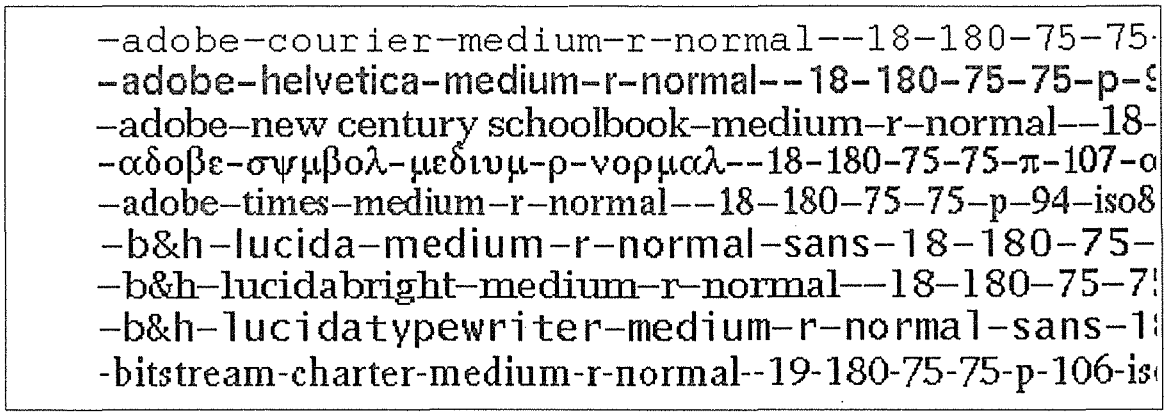
When you think of the X fonts as comprising several large font families rather than as hundreds of individual (unique) fonts, you can quickly reduce the clutter. Figure 6-2 shows the major families of commercial fonts that are available under X. To illustrate the fonts, we've used the simple expedient of printing each font name in the font itself. Font names are truncated to fit on the page.† (For those of you who don't read the Greek alphabet, the fourth line down reads "-adobe-symbol-medium-r-normal--18 ..." This font is used for mathematical equations and so forth, rather than for normal display purposes.) You'll notice that with the exception of Courier and Lucidatypewriter, all of the fonts in the figure are *proportionally spaced*. That is, each character has a different width. This makes them look good on a printed page but makes them less appropriate for screen display in terminal windows (especially for program editing), since text will not line up properly unless all characters are the same width.

You will most likely use these proportional fonts for labels or menu items, rather than for running text. (Word processing or publishing programs will, of course, use them to represent proportional type destined for the printed page.)

Courier and Lucidatypewriter are *monospaced*, which means that every character has the same width. Although monospaced fonts *can* be used for the text font in *xterm* windows, if you do so, you may notice that some "garbage" pixels will occasionally be left on the screen. This effect happens because the characters are not clearly contained or divided from one another.

*By contrast, the Macintosh supports dozens of font families; commercial typesetters support hundreds and, in some cases, even thousands of families. Many of these fonts will doubtless be made commercially available for X.

†To generate the figures in this section and in Appendix B, *Release 5 Standard Fonts*, we wrote a short program called *xshowfonts*, which displays a series of fonts in a scrollable window. Source code for *xshowfonts* is listed in Appendix B, *Release 5 Standard Fonts*. In each case, we used wildcards (discussed later in this chapter) to select the fonts we wanted and then did screendumps of the resulting images. Note that the fonts look better on the screen than they do in the illustration, since the scaling factor used to make the screen dumps exacerbates the "jagged edges" endemic to bitmap fonts.



```
-adobe-courier-medium-r-normal--18-180-75-75
-adobe-helvetica-medium-r-normal--18-180-75-75-p-
-adobe-new century schoolbook-medium-r-normal--18-
-αδοβε-σψμβολ-μεδιυμ-ρ-νορμαλ--18-180-75-75-π-107-ο
-adobe-times-medium-r-normal--18-180-75-75-p-94-iso8
-b&h-lucida-medium-r-normal-sans-18-180-75-
-b&h-lucidabright-medium-r-normal--18-180-75-7
-b&h-lucidatypewriter-medium-r-normal-sans-1
-bitstream-charter-medium-r-normal--19-180-75-75-p-106-is
```

Figure 6-2. The major commercial font families available in the standard X distribution

For *xterm* and other terminal emulators, you're better off using *character cell fonts*. These fonts are special monospaced fonts originally designed for computer displays. Each character in a monospaced font has the same width. Character cell fonts go a bit further in that an invisible cell contains every character. The spacing relates to the size of the cell that contains each character, rather than to the character itself.

As you may recall, these character cell fonts have simple aliases that correspond to their dimensions in pixels. For example, in the font named 8x13, each character occupies a box 8 pixels wide by 13 pixels high. (To fit the logical font naming conventions, these fonts have been given a foundry name of "misc" and a font family of "fixed.") There are also one or two larger fixed fonts donated by Sony for use with their extra-high resolution monitor, with a foundry name of "sony." Figure 6-3 shows some of the character cell fonts, using their aliases in R5.

Table 6-1 shows the correspondence between these aliases and full font names. Note that the 6x13 font also has an additional alias called "fixed" defined for it. The "fixed" alias is used as the default font for *xterm* windows. (Twelve-point Helvetica bold roman has the alias "variable," which several applications use as the default font for labels. *mwm* uses this font for the application name that appears in the title area of the window frame.)

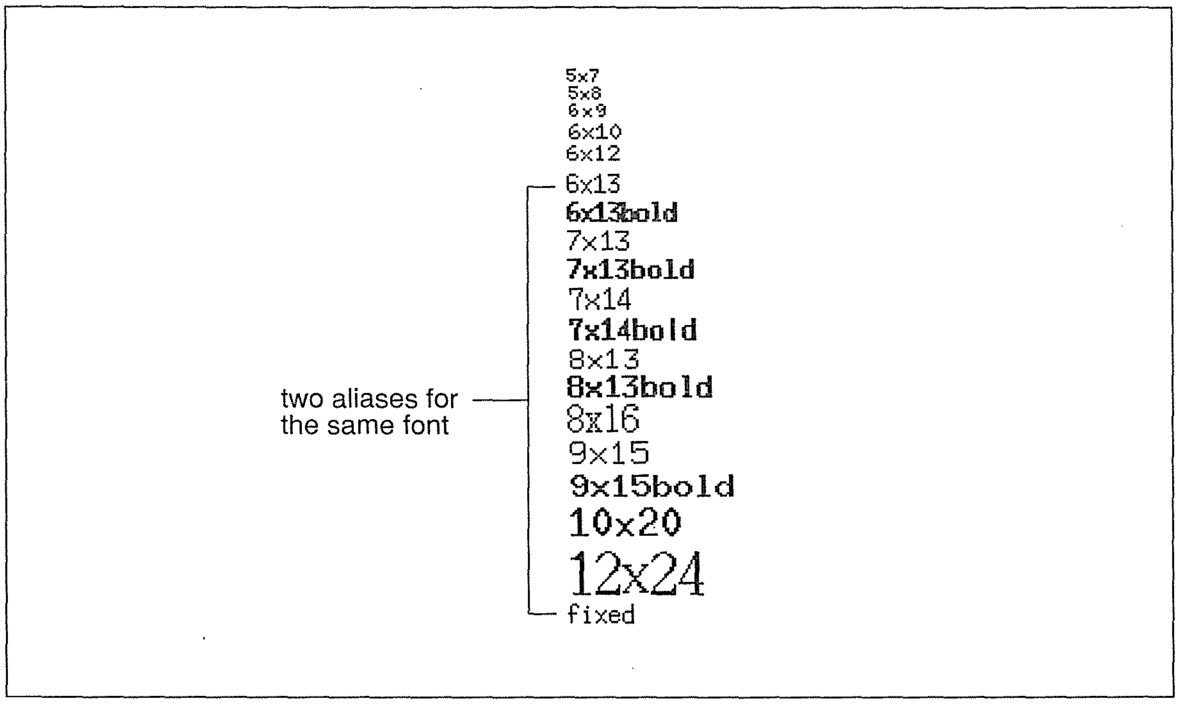


Figure 6-3. Miscellaneous fonts for xterm text

Table 6-1. Fixed Font Aliases and Font Names

Alias	Filename
fixed	-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
5x7	-misc-fixed-medium-r-normal--7-70-75-75-c-50-iso8859-1
5x8	-misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-1
6x9	-misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
6x10	-misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
6x12	-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
6x13	-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
6x13bold	-misc-fixed-bold-r-semicondensed--13-120-75-75-c-60-iso8859-1
7x13	-misc-fixed-medium-r-normal--13-120-75-75-c-70-iso8859-1
7x13bold	-misc-fixed-bold-r-normal--13-120-75-75-c-70-iso8859-1
7x14	-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
7x14bold	-misc-fixed-bold-r-normal--14-130-75-75-c-70-iso8859-1
8x13	-misc-fixed-medium-r-normal--13-120-75-75-c-80-iso8859-1
8x13bold	-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
8x16	-sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-1

Table 6-1. *Fixed Font Aliases and Font Names (continued)*

Alias	Filename
9x15	-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
9x15bold	-misc-fixed-bold-r-normal--15-140-75-75-c-90-iso8859-1
10x20	-misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1
12x24	-sony-fixed-medium-r-normal--24-170-100-100-c-120-iso8859-1

R5 also includes the Clean family of fixed-width fonts from Schumacher, and DEC's terminal fonts, both of which are illustrated in Appendix B, *Release 5 Standard Fonts*.

There are also many other special purpose fonts, such as the Greek Symbol font that we already saw, the cursor font, the OPEN LOOK™ cursor and glyph fonts, Hangul fonts, Hebrew fonts, and Kana and JIS Kanji Japanese fonts.

As mentioned previously, Bitstream, Inc. has donated a few Charter and Courier outline fonts. For more information, see "The Size of Bitmap and Outline Fonts" and "Specifying Scalable Fonts" later in this chapter.

See Appendix B, *Release 5 Standard Fonts*, for comprehensive lists of all standard fonts, as well as pictures of the complete character set in some representative fonts.

Stroke Weight and Slant

The characters in a given font family can be given a radically different appearance by changing the *stroke weight* or the *slant*, or both.

The most common weights are medium and bold. The most common slants are roman (upright), italic, or oblique. (Both italic and oblique are slanted; however, italic versions of a font generally have had the character shape changed to make a more pleasing effect when slanted, while oblique fonts are simply a slanted version of the upright font. In general, *serif* fonts (those with little decorations on the ends and corners of the characters) are slanted via italics, while *sans-serif* fonts are made oblique.) (Whether a font is sans serif comes under the category of additional style; see "Other Information in the Font Name.")

Figure 6-4 compares the medium and bold weights, and the roman and italic or oblique slants in the Charter® and Helvetica font families.

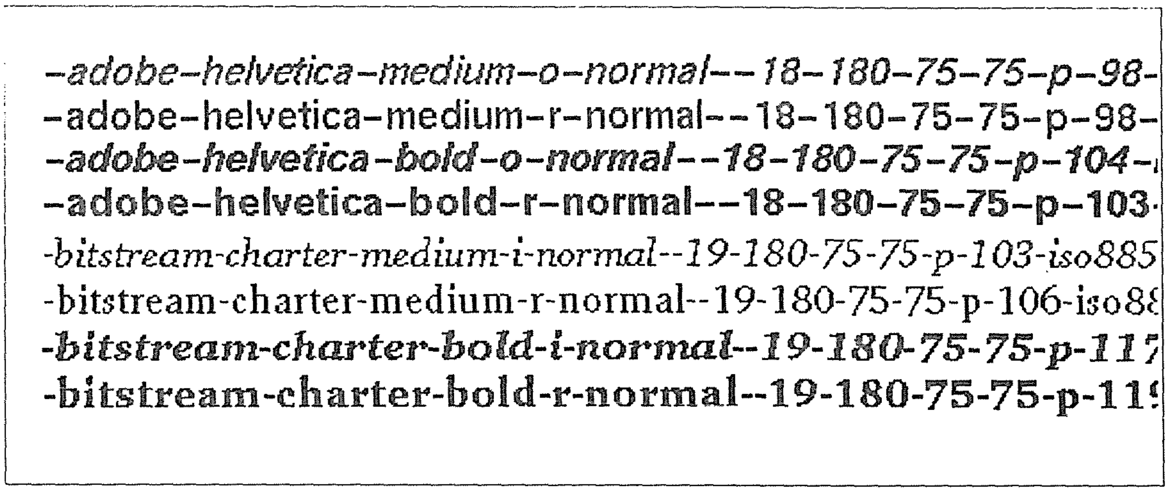

-adobe-helvetica-medium-o-normal--18-180-75-75-p-98-
-adobe-helvetica-medium-r-normal--18-180-75-75-p-98-
-adobe-helvetica-bold-o-normal--18-180-75-75-p-104-
-adobe-helvetica-bold-r-normal--18-180-75-75-p-103-
-bitstream-charter-medium-i-normal--19-180-75-75-p-103-iso885
-bitstream-charter-medium-r-normal--19-180-75-75-p-106-iso8
-bitstream-charter-bold-i-normal--19-180-75-75-p-11
-bitstream-charter-bold-r-normal--19-180-75-75-p-11

Figure 6-4. The same fonts in different weights and slants

Release 5 also includes one font that has an in-between weight called *demibold*. Weight names are somewhat arbitrary, since a demibold weight in one family may be almost as dark as a bold weight in another.

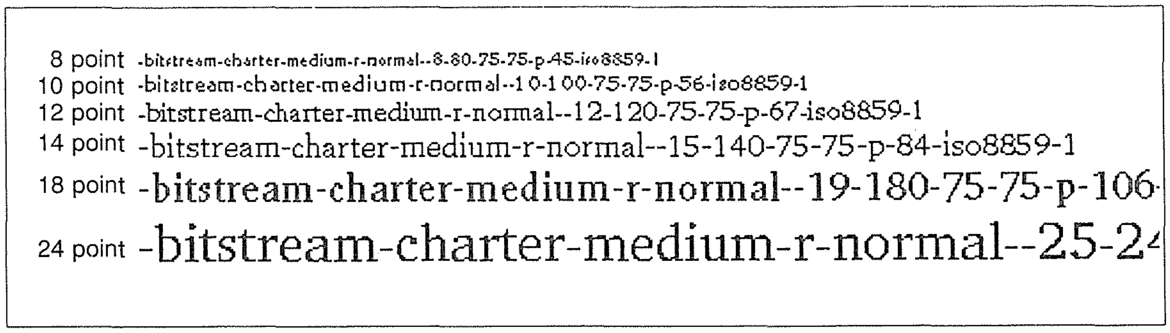
The font naming convention also defines two counter-clockwise slants called *reverse italic* (ri) and *reverse oblique* (ro), as well as a catch-all called *other* (ot).

The Size of Bitmap and Outline Fonts

Font sizes are often given in a traditional printer's measure known as a *point*. A point is approximately one seventy-second of an inch. You'll probably need to look at some fonts to get a better idea of point sizes, but most typewriters produce characters in either 10 or 12 point type. The characters in X fonts are measured both in points and in *pixels*, which can also be thought of as *dots*, or the individual picture elements that make up an image.

As mentioned previously (in "What's New in Release 5?"), the standard distribution of X includes two types of fonts: *bitmap fonts* and *outline fonts*. Outline, or scalable, fonts are intended to be scaled to any size you request. Thus, outline font names have zeros in the columns for point and pixel size. You normally request a particular size by filling in a point size. The most important thing to remember about these scalable outline fonts is that they are device-independent and, thus, true-to-size regardless of the screen. The section "Specifying Scalable Fonts" later in this chapter explains how to get the size font you want.

Most of the fonts provided with X are bitmap fonts. Though bitmap fonts *can* be scaled, they are not intended to be and scaled versions of bitmap fonts may have a jagged appearance. Each size and orientation of a bitmap font must be stored in a separate file. Most of the bitmap font families are provided in the six point sizes shown in Figure 6-5.



8 point -bitstream-charter-medium-r-normal--8-80-75-75-p-45-iso8859-1
10 point -bitstream-charter-medium-r-normal--10-100-75-75-p-56-iso8859-1
12 point -bitstream-charter-medium-r-normal--12-120-75-75-p-67-iso8859-1
14 point -bitstream-charter-medium-r-normal--15-140-75-75-p-84-iso8859-1
18 point -bitstream-charter-medium-r-normal--19-180-75-75-p-106-
24 point -bitstream-charter-medium-r-normal--25-2

Figure 6-5. The same font in six different point sizes

Font size is not much of an issue when you're dealing with outline fonts. However, because of the different resolution of computer monitors, a bitmap font with a given point size might actually appear larger or smaller, depending on the particular screen. In the remainder of this section, we'll discuss the various factors that determine how large a font appears on your screen. This information might seem somewhat arcane, but it will be relevant to those dealing extensively with fonts. Everyone else should skip ahead.

Resolution is often spoken about in terms of the screen's dimensions in pixels. The *xdpyinfo* client, described in Chapter 8, *Other Clients*, will tell you this information; your workstation or X terminal documentation may also provide it. *xdpyinfo* also tells you the screen's resolution in dots per inch. Most monitors on the market today have a resolution between 75 dots per inch (dpi) and 100 dots per inch.

Accordingly, there are both 75-dpi and 100-dpi versions of most of the bitmap fonts in the standard distribution. These separate versions of each font are stored in different directories. By setting the font search path so that the appropriate directory comes first, you can arrange to get the correct versions without having to specify them in the font name.* Keep in mind that monitors can vary substantially. The Sun 19-inch monitor has a resolution of 1152 x 900 pixels (approximately 90 dpi). The NCD 16-inch X terminal has a resolution of 1024 x 1024 pixels (approximately 106 dpi).

What happens if you select the wrong resolution for your monitor? Given the difference in the pixel size, the same size font will appear larger or smaller than the nominal point size.

For example, consider the 75- and 100-dpi versions of the 24-point charter medium italic font:

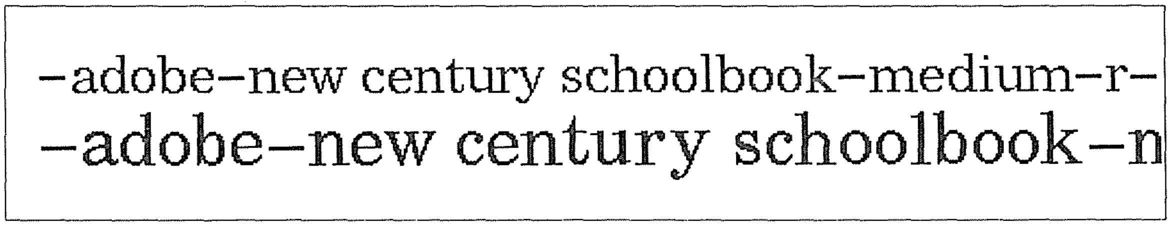
```
-bitstream-charter-medium-i-normal--25-240-75-75-p-136-iso8859-1
-bitstream-charter-medium-i-normal--33-240-100-100-p-179-iso8859-1
```

If you look at the pixel size field, you will notice that the height of the 75-dpi version is 25 pixels, while the height of the 100-dpi version is 33 pixels. If you use the 75-dpi version on the Sun, you actually get something closer to 20 points (75/90*24); on a 100-dpi monitor, you will actually get something closer to 18 points (75/100*24). We noticed this right away when we first began using the NCD X terminal. Because of the NCD's higher resolution, the font size we had been using on the Sun appeared smaller.

*We'll talk about how to set the font search path later in this chapter.

If you are working on a lower-resolution monitor, you can take advantage of this artifact to display type as large as 32 points (the size that a 24-point 100-dpi font will appear on a 75-dpi monitor.) Figure 6-6 shows the 75- and 100-dpi versions of the same 24-point font, as displayed on a Sun workstation with a 19-inch monochrome monitor. As shown, neither is actually 24 points. The 75-dpi version is actually 20 points, as discussed above; the 100-dpi version is about 26.4 points.*

Note that the logical font-naming convention allows for different horizontal and vertical resolution values. This would allow server manufacturers to support fonts that were "tuned" for their precise screen resolution. However, the bitmap fonts shipped with the generic X11 distribution all use the same horizontal and vertical resolution.



```
-adobe-new century schoolbook-medium-r-
-adobe-new century schoolbook-n
```

Figure 6-6. The 100-dpi version of a 24-point font appears larger on a 75-dpi monitor

As suggested above, this resolution may not exactly match the actual resolution of any particular screen, resulting in characters that are not true to their nominal point size. On the Sun monitor, neither the 75- nor 100-dpi fonts will be right. (Of course, if you are using scalable outline fonts, this isn't a problem.)

Other Information in the Font Name

What we've already shown summarizes the most important information in the font name. The remaining fields are explained below:

Foundry — Font manufacturers are still referred to as foundries, from the days when type was cast from lead. The X font naming convention specifies the foundry as the company that digitized or last modified the font, rather than its original creator.

For the fonts contained in the standard X distribution, the foundry is not terribly significant since there are no cases where the same font family is available from different foundries. However, there are numerous commercial font families available from more than one foundry. In general, the appearance of the fonts should be quite similar since the font family defines the design of the typeface. However, there may be some small differences

*Note that the differences are exaggerated further in printing the screen dump of this display. *xpr* lets you select a scale factor such that each pixel on the screen appears as *scale* pixels in the printout. Since the laser printer has a 300-dpi resolution, a scale factor of 4 would produce a true scale screen dump if the resolution on the Sun monitor were truly 75 dpi by 75 dpi. Since it is actually 90 by 90, the printed image is enlarged by about 16 percent.

	in the quality of some of the characters, and there may be more significant differences in the font metrics (the vertical or horizontal measurements of the characters). This might be significant for a publishing application that was using the bitmapped font for a *wysiwyg** screen display that needed to match the fonts in a particular laser printer or typesetter.
Set width	A value describing a font's proportionate width, according to the foundry. Typical set widths include: normal, condensed, semicondensed, narrow, double width. Most of the Release 5 fonts have the set width *normal.* A few fonts have the set width *semicondensed.*
Additional Style	Not represented in most R5 font names. However, according to the logical font convention, the style of a font may be specified in the field between set width and pixels. Some of the possible styles are *i* (informal), *r* (roman), *serif* and *sans* (serif). Currently all of the standard fonts have *sans* or an empty field. Note that the *r* for roman may also be used in the slant field.
Spacing	All standard Release 5 fonts are either: *p* (proportional, i.e., variable-width); *m* (monospace, i.e., fixed-width); or *c* (character cell, a fixed-width font based on the traditional typewriter model, in which each character can be thought to take up the space of a "box" of the same height and width).
Average width	Mean width of all characters in the font, measured in tenths of a pixel. You'll notice, if you look back at Figure 6-2, that two fonts with the same point size (such as New Century Schoolbook and Times) can have a very different average character width. This field can sometimes be useful if you are looking for a font that is especially wide or especially narrow.
	The Schumacher Clean family of fonts offers several fonts in the same point size but with different average widths.†
Character set	In the initial illustration of the font naming convention, Figure 6-1, we identified the character set as a single field. If you look more closely, you'll realize it is actually two fields, the first of which identifies the organization or standard registering the character set, the second of which identifies the actual character set.
	Most fonts in the standard X distribution contain the string "iso8859-1" in their names, which represents the ISO Latin-1 character set. The ISO Latin-1 character set is a superset of the standard ASCII character set, which includes various special characters used in European languages other than English. See Appendix H of Volume Two, *Xlib Reference Manual*, for a complete listing of the characters in the ISO Latin-1 character set.

*This is an acronym for "what you see is what you get" and describes a type of text editor or word processor that purports to display the page exactly as it would appear in print. There is some approximation due to the differences between screen fonts and printer fonts. MacWrite® is a *wysiwyg* program in the MacIntosh world; FrameMaker, Interleaf and others offer wysiwyg programs in the X world.

†These fonts all (incorrectly to our minds) have a set width of "normal." They should be distinguished by set widths such as condensed, semi-condensed, etc. Since they do not, they can be distinguished by the difference in their average width.

Note, however, that the symbol font contains the strings "adobe-fontspecific" in this position. This means that Adobe Systems defined the character set in this font, and that it is font-specific. You can see from this example that the use of these fields is somewhat arbitrary.

For a complete technical description of font naming conventions, see the X Consortium Standard, *X Logical Font Description Conventions*. This document is available as part of the standard MIT X distribution, and is reprinted as Appendix M in the second edition of Volume 0, *X Protocol Reference Manual*.

Font Name Wildcarding

To simplify font specification, X supports the use of wildcards within font names. An asterisk (*) can be used to represent any part of the font name string; a question mark (?) can be used to represent any single character. You can usually get the font you want by specifying only the font family, the weight, the slant, and the point size, and wildcarding the rest. For example, to get Courier bold at 14 points, you could use the command-line option:

```
-fn '*courier-bold-r*140*'
```

That's starting to seem a little more intuitive!

However, there are a number of "gotchas."

- First, since the UNIX C shell also has a special meaning for the * and ? wildcard characters, wildcarded font names must be quoted. This can be done by enclosing the entire font name in quotes (as in the previous example), or by "quoting" each wildcard character by typing a backslash before it. (If you don't do this, the shell will try to expand the * to match any filenames in the current directory, and will give the message "No match.") Wildcards need not be quoted in resource files.
- Second, if the wildcarded font name matches more than one font, the server will use the first one that matches. And unfortunately, because the names are sorted in simple alphabetical order, the bold weight comes before medium, and italic and oblique slants before roman. As a result, the specification:

  ```
  -fn '*courier*'
  ```

  will give you Courier bold oblique, rather than the Courier medium roman you might intuitively expect.

  If you aren't sure whether your wildcarded name is specific enough, try using it as an argument to *xlsfonts*. If you get more than one font name as output, you may not get what you want. Try again with a more specific name string.

  The exception to this rule has to do with the *75dpi* and *100dpi* directories. If a wildcard matches otherwise identical fonts in these two directories, the server will actually use the one in the directory that comes first in the font path. This means that you should put the appropriate directory first in the font path. (We'll tell you how to do this in the next section.) Thereafter, you can generally wildcard the resolution fields (unless you specifically want a font from the directory later in the path).*

- Third, the asterisk (*) wildcard expansion is resolved by a simple string comparison. So, for example, if you type:

  ```
  -fn '*courier-bold*r*140*'
  ```

  instead of:

  ```
  -fn '*courier-bold-r*140*'
  ```

  (the difference being the asterisk instead of the hyphen before the "r" in the slant field), the "r" would also match the "r" in the string "normal" in the set width field. The result is that you would select all slants. Since o (oblique) comes before r (roman), and you always get the first font that matches, you'd end up with Courier oblique.

  The trick is to be sure to include at least one of the hyphens to set the -r- off as a separate field rather than as part of another string.

  Even though a wildcarded name such as:

  ```
  *cour*b*r-*140*
  ```

  should get you 14-point Courier bold roman, we think it is good practice to spell out the font family and weight and use hyphens between adjacent fields. As usual there are exceptions: the Lucida family really has three subfamilies; you can get all three by specifying the family as "Lucida*" rather than "Lucida-"; and you might certainly want to abbreviate "New Century Schoolbook" to "New Century*" or "*Schoolbook."

- Font names are case-insensitive. "Courier" is the same as "courier."

Table 6-2 summarizes the values you should use to specify a font name (assuming only the standard fonts are loaded). Choose one element from each column. Don't forget to include the leading and trailing asterisks and the hyphen before the slant.

*Unlike *xfontsel*, which displays fonts in the order of wildcard matches, *xlsfonts* will always list fonts in straight-sort order, with the sort done character by character across the line. Since size in pixels comes before point size in the name, and the size in pixels of the 100-dpi fonts is larger than that of the equivalent 75-dpi font, the 75-dpi font will always be listed first for a given point size. But when listing more than one point size, the fonts will be jumbled. For example, the size in pixels of the 8-point Charter font at 100-dpi is 11, so it will come after the 10-point Charter font at 75 dpi, with a size in pixels of 10. The 8-point Charter font at 75 dpi gets sorted to the very end of the list, since to a character-by-character sort, its size in pixels (8) looks larger to the size in pixels of even the largest 100-dpi font (the 24 point, with a height of 33 pixels).

Table 6-2. Essential Elements of a Font Name

*	Family	–	Weight	–	Slant	*	Point Size	*
	charter		medium		r (roman)		80 (8 pt.)	
	clean		bold		i (italic)		100 (10 pt.)	
	courier		demibold		o (oblique)		120 (12 pt.)	
	fixed				ri (reverse italic)		140 (14 pt.)	
	gothic				ro (reverse oblique)		180 (18 pt.)	
	helvetica				ot (other)		240 (24 pt.)	
	lucida							
	lucidabright							
	lucidatypwriter							
	mincho							
	new century schoolbook							
	nil							
	open look cursor							
	open look glyph							
	symbol							
	terminal							
	times							

Note that the point sizes listed in the table correspond to the point sizes of the standard bitmap fonts in the *75dpi* and *100dpi* directories. If you are specifying one of the scalable outline fonts, you do not have to limit yourself to these sizes. You can also specify alternative sizes for the bitmap fonts, but the results may not be very legible. See "Previewing and Selecting Fonts: xfontsel" for information about previewing fonts.

Note also that if you specify either Charter or Courier in a medium or bold weight, with either a roman or italic slant, you will get one of the scalable outline fonts in the *Speedo* directory before an analogous bitmap font in the *75dpi* directory. (This assumes that the default font search path is in effect.) You can avoid this issue with Courier fonts by additionally providing the foundry name ("adobe" for the bitmap version, "bitstream" for the outline version). Bitstream provides analogous Charter fonts in the *Speedo*, *75dpi*, and *100dpi* directories, so the font path is critical. See "The Font Search Path" and "Specifying Scalable Fonts" for more information.

Another way to avoid these issues is to provide unique aliases for all bitmap fonts. See "Font Name Aliasing" later in this chapter for more information.

Specifying Scalable Fonts

The *Speedo* directory in the standard distribution provides eight scalable outline fonts, whose names are listed below:

```
-bitstream-charter-medium-r-normal--0-0-0-0-p-0-iso8859-1
-bitstream-charter-medium-i-normal--0-0-0-0-p-0-iso8859-1
-bitstream-charter-bold-r-normal--0-0-0-0-p-0-iso8859-1
-bitstream-charter-bold-i-normal--0-0-0-0-p-0-iso8859-1
-bitstream-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
-bitstream-courier-medium-i-normal--0-0-0-0-m-0-iso8859-1
-bitstream-courier-bold-r-normal--0-0-0-0-m-0-iso8859-1
-bitstream-courier-bold-i-normal--0-0-0-0-m-0-iso8859-1
```

Notice that the three fields relating to size (pixels, points, and average width), plus the two resolution fields, have zeroes in them. You can specify any one of these scalable outline fonts simply by adding a point size. The following name scales the Courier bold italic font to 24 points (on most screens, a huge font):

```
-bitstream-courier-bold-i-normal--0-240-0-0-m-0-iso8859-1
```

Note that you can also scale any of the bitmap fonts (to a different size than those offered) by taking the font name and:

1. Changing the pixel and average width fields to zero;
2. Specifying the point size you want (in tenths of a point).

Note that every field must have something in it (even if you use an asterisk wildcard). Don't use an asterisk to replace multiple adjacent fields. The following name scales the Adobe Courier bold font (roman slant) to 15 points (a size not usually available).

```
-adobe-courier-bold-r-normal--0-150-75-75-m-0-iso8859-1
```

Remember, however, that scaling bitmap fonts may produce "jagged" results.

The Font Search Path

In Release 5, fonts are stored in four directories, as shown in Table 6-3.

Table 6-3. Standard Font Directories, Release 5

Directory	Contents
/usr/lib/X11/fonts/misc	Fixed-width fonts suitable for terminal emulation, the standard cursor font, several Clean family fonts provided by Schumacher, two JIS Kanji fonts, several Kana fonts from Sony Corporation, two Hangul fonts from Daewoo Electronics, two Hebrew fonts from Joseph Friedman, two cursor fonts from Digital Equipment Corporation, and OPEN LOOK cursor and glyph fonts from Sun Microsystems, Inc.
/usr/lib/X11/fonts/Speedo	Charter and Courier (scalable) outline fonts donated by Bitstream, Inc. (New in Release 5.)
/usr/lib/X11/fonts/75dpi	Fixed- and variable-width fonts, 75 dpi, contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc.
/usr/lib/X11/fonts/100dpi	Fixed- and variable-width fonts, 100 dpi, many the 100 dpi versions of the fonts in the *75dpi* directory.

These four directories (in this order) constitute X's default font path.

Modifying the Font Search Path

You can change the font path using the *xset* client with the `fp` (font path) option. *xset* allows you to:

- Rearrange the order of the directories in the font search path;
- Add directories to or substract directories from the font path;
- Completely replace the font path.

For example, say you add fonts to a directory called *newfonts*, a subdirectory of your home directory.* To access these fonts, you add the directory to the font path (in effect, inform the X server about the directory).

*Whenever you add fonts to a directory (according to the guidelines in Volume Eight), before running *xset* you must first run the command:

```
% mkfontdir directory_name
```

Volume Eight, *X Window System Administrator's Guide*, explains how to convert, compile, and install fonts.

Use *xset* with the fp+ option to add a directory or list of directories to the end of the font path. The following command appends the *~/newfonts* directory* to the path:

```
% xset fp+ ~/newfonts
```

We can verify that the new path is in effect by running *xset* with the q option (which "queries" the settings):

```
% xset q
      .
      .
      .
Font Path:
/usr/lib/X11/fonts/misc/,/usr/lib/X11/fonts/Speedo/,/usr/lib/X11/fonts/75dpi/,\
/usr/lib/X11/fonts/100dpi/,~/newfonts
```

(The preceding output is broken onto two lines escaped with a backslash (\) only so that it can be printed within the page margins.)

Notice that in adding directories to the font search path, the location of the plus sign is significant. The command:

```
% xset +fp ~/newfonts
```

adds *~/newfonts* to the beginning of the existing path.

You can remove a directory from the font path using either the -fp or fp- option. The location of the minus sign is not significant. To remove *~/newfonts* from the font path, enter:

```
% xset -fp ~/newfonts
```

or:

```
% xset fp- ~/newfonts
```

Note that you can add or subtract multiple directories from the font path with a single invocation of *xset*. After the relevant fp option, the argument should be a comma-separated list of directories, as in the command:

```
% xset fp+ ~/newfonts,/usr/lib/X11/fonts/new
```

This command adds the directories *~/newfonts* and */usr/lib/X11/fonts/new* to the end of the current font path.

To completely replace the font path (rather than append to or subtract from the current path), use fp= followed by a comma-separated list of directories:

```
% xset fp= fontdir1,fontdir2,fontdir3,...
```

Note that a space must follow the equal sign (=).

You would also use xset fp= to change the order of directories in the current path. For example, to put the *100dpi* directory before the *75dpi* directory, you would enter:

```
% xset fp= /usr/lib/X11/fonts/misc,/usr/lib/X11/fonts/Speedo,\
                /usr/lib/X11/fonts/100dpi,/usr/lib/X11/fonts/75dpi
```

You can restore the default font path at any time by entering:

```
% xset fp default
```

Note that if you want to access fonts via a remote (or even a local) font server (*fs*) program, you must add the font server to the path. See "Using the Font Server" later in this chapter for more information.

The fonts.dir Files

In addition to font files, each font directory contains a file called *fonts.dir*. The *fonts.dir* files serve, in effect, as databases for the X server. When the X server searches the directories in the default font path, it uses the *fonts.dir* files to locate the font(s) it needs.

Each *fonts.dir* file contains a list of all the font files in the directory with their associated font names in two-column form. (The first column lists the font file name and the second column lists the actual font name associated with the file.) The first line in *fonts.dir* lists the number of entries in the file (i.e., the number of fonts in the directory).

Example 6-1 shows a portion of the *fonts.dir* file from the Release 5 */usr/lib/X11/fonts/100dpi* directory. As the first line indicates, the directory contains 200 fonts. The first group of fonts listed below (up to the second ellipse) are all Courier family fonts. The second group of fonts shown in the list below are a few sizes from the Charter family.

Example 6-1. Subsection of the Release 5 fonts.dir file in /usr/lib/X11/fonts/100dpi

```
200
 .
 .
 .
courBO08.snf -adobe-courier-bold-o-normal--11-80-100-100-m-60-iso8859-1
courBO10.snf -adobe-courier-bold-o-normal--14-100-100-100-m-90-iso8859-1
courBO12.snf -adobe-courier-bold-o-normal--17-120-100-100-m-100-iso8859-1
courBO14.snf -adobe-courier-bold-o-normal--20-140-100-100-m-110-iso8859-1
courBO18.snf -adobe-courier-bold-o-normal--25-180-100-100-m-150-iso8859-1
courBO24.snf -adobe-courier-bold-o-normal--34-240-100-100-m-200-iso8859-1
courB08.snf -adobe-courier-bold-r-normal--11-80-100-100-m-60-iso8859-1
courB10.snf -adobe-courier-bold-r-normal--14-100-100-100-m-90-iso8859-1
courB12.snf -adobe-courier-bold-r-normal--17-120-100-100-m-100-iso8859-1
courB14.snf -adobe-courier-bold-r-normal--20-140-100-100-m-110-iso8859-1
courB18.snf -adobe-courier-bold-r-normal--25-180-100-100-m-150-iso8859-1
courB24.snf -adobe-courier-bold-r-normal--34-240-100-100-m-200-iso8859-1
courO08.snf -adobe-courier-medium-o-normal--11-80-100-100-m-60-iso8859-1
courO10.snf -adobe-courier-medium-o-normal--14-100-100-100-m-90-iso8859-1
courO12.snf -adobe-courier-medium-o-normal--17-120-100-100-m-100-iso8859-1
courO14.snf -adobe-courier-medium-o-normal--20-140-100-100-m-110-iso8859-1
courO18.snf -adobe-courier-medium-o-normal--25-180-100-100-m-150-iso8859-1
courO24.snf -adobe-courier-medium-o-normal--34-240-100-100-m-200-iso8859-1
courR08.snf -adobe-courier-medium-r-normal--11-80-100-100-m-60-iso8859-1
courR10.snf -adobe-courier-medium-r-normal--14-100-100-100-m-90-iso8859-1
courR12.snf -adobe-courier-medium-r-normal--17-120-100-100-m-100-iso8859-1
courR14.snf -adobe-courier-medium-r-normal--20-140-100-100-m-110-iso8859-1
courR18.snf -adobe-courier-medium-r-normal--25-180-100-100-m-150-iso8859-1
courR24.snf -adobe-courier-medium-r-normal--34-240-100-100-m-200-iso8859-1
 .
 .
 .
```

Example 6-1. Subsection of the Release 5 fonts.dir file in /usr/lib/X11/fonts/100dpi (continued)

```
charBI08.snf -bitstream-charter-bold-i-normal--11-80-100-100-p-68-iso8859-1
charBI10.snf -bitstream-charter-bold-i-normal--14-100-100-100-p-86-iso8859-1
charBI12.snf -bitstream-charter-bold-i-normal--17-120-100-100-p-105-iso8859-1
charBI14.snf -bitstream-charter-bold-i-normal--19-140-100-100-p-117-iso8859-1
charBI18.snf -bitstream-charter-bold-i-normal--25-180-100-100-p-154-iso8859-1
charBI24.snf -bitstream-charter-bold-i-normal--33-240-100-100-p-203-iso8859-1
  .
  .
  .
```

The *fonts.dir* files are created by the *mkfontdir* client when X is installed. In the case of bitmap fonts, *mkfontdir* reads the font files in the relevant directories in the path, extracts the font names, and creates a *fonts.dir* file in each directory. Scalable outline fonts (in the *Speedo* directory) require an intermediary file called *fonts.scale*, which *mkfontdir* converts to a *fonts.dir* file. The standard distribution includes a *fonts.scale* file in the *Speedo* directory.

If *fonts.dir* files are present on your system, you probably won't have to deal with them or with *mkfontdir* at all. If the files are not present, or if you have to load new fonts or remove existing ones, you will have to create files with *mkfontdir*. Refer to the *mkfontdir* reference page in Part Three of this guide and Volume Eight, *X Window System Administrator's Guide*, for more information.

Font Name Aliasing

Another way to abbreviate font names is by aliasing (that is, by associating fonts with alternative names of your own choosing). You can edit or create a file called *fonts.alias*, in any directory (or multiple directories) in the font search path, to set aliases for existing fonts. The X server uses both *fonts.dir* files and *fonts.alias* files to locate fonts in the font path.

Release 5 provides a default *fonts.alias* file for each of the three bitmap font directories (*misc*, *75dpi*, and *100dpi*). (The scalable outline fonts in the *Speedo* directory cannot be aliased because you must provide a point size!) Take the time to look at the contents of each of these files, since many of the existing aliases may be easier to type than even wildcarded font names. If you have write permission for the directory, you can also add aliases to the file, change existing aliases, or even replace the entire file. However, this should be done with caution. To play it safe, it's probably a good idea merely to *add* to existing *fonts.alias* files. If you're working in a multiuser environment, the system administrator should definitely be consulted before aliases are added or changed. Note that when you create or edit a *fonts.alias* file, the server does not *automatically* recognize the aliases in question. You must make the server aware of newly created or edited alias files by resetting the font path with *xset*.

The *fonts.alias* file has a two-column format similar to the *fonts.dir* file: the first column contains aliases, the second column contains the actual font names. If you want to specify an alias that contains spaces, enclose the alias in double quotes. If you want to include double quotes (") or other special characters as part of an alias, precede each special symbol with a backslash (\).

When you use an alias to specify a font in a command line, the server searches for the font associated with that alias in every directory in the font path. Therefore, a *fonts.alias* file in one directory can set aliases for fonts in other directories as well. You might choose to create a single aliases file in one directory of the font path to set aliases for the most commonly used fonts in all the directories. Example 6-2 shows three sample entries that could be added to an existing *fonts.alias* file (or constitute a new one).

Example 6-2. Sample fonts.alias file entries

```
cour12     -adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
cour14     -adobe-courier-medium-r-normal--14-140-75-75-m-90-iso8859-1
cour18     -adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1
```

As the names of the aliases suggest, these sample entries provide aliases for three Courier fonts of different point sizes. You can also use wildcards within the font names in the right column of an alias file. For instance, the alias file entries above might also be written as follows:

```
cour12     *courier-medium-r-*-120*
cour14     *courier-medium-r-*-140*
cour18     *courier-medium-r-*-180*
```

If you would like to be able to use font filenames as aliases, you must explicitly assign every font name an alias corresponding to the name of the file in which it is stored. (It's a good idea to use the filename without the *.pcf* extension.) This could actually be done rather easily by editing a copy of each *fonts.dir* file and appending the copy to the *fonts.alias* file in the same directory. (These aliases could also be appended to the *fonts.alias* file in the *misc* directory, since the server searches all directories in the font path.)

Once the server is made aware of aliases, you can specify an alias on the command line. For example, you can use a font name alias as an argument to *xfd*. If you've used an alias file or files to specify font names (minus extensions) as aliases, you can display the font stored in the file *courR12.pcf* using the command:

```
    % xfd -fn courR12 &
```

A special note about the *misc* directory: when X was configured for your system, a *fonts.alias* file should have been created in this directory. The first two entries in this file are shown below:

```
fixed    -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
variable -*-helvetica-bold-r-normal-*-*-120-*-*-*-*-iso8859-1
```

The default file contains an additional 64 entries but the entries pictured above are particularly important. The aliases called "fixed" and "variable" are invoked as the default fonts for many clients. The "fixed" font can be thought of as a system-wide default. The "variable" font, described in the right column as a 12-point bold Helvetica font, is used as the default font by *bitmap*, as well as by other clients. If this file is removed or replaced, when you run *bitmap*, you'll get an error message that the server cannot open the variable font, and text in the *bitmap* window will display in the smaller, somewhat less readable "fixed" font.

If you do choose to edit the *fonts.alias* file in the *misc* directory, it is important to preserve at least these two aliases. As we've said, it's probably a better idea to keep all the default entries and merely append any new ones.* Regardless of what edits you make to the file, the line specifying the variable alias must not be changed.

Making the Server Aware of Aliases

After you create (or update) an alias file, the server does not automatically recognize the aliases in question. You must make the server aware of newly created or edited alias files by "rehashing" the font path with *xset*. Enter:

```
% xset fp rehash
```

on the command line. The *xset* option `fp` (font path) with the `rehash` argument 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. (You also need to use *xset* if you want to change the font path. See "Modifying the Font Search Path" earlier in this chapter for details.)

Utilities for Displaying Information about Fonts

We've already mentioned *xlsfonts*, which simply displays the names and aliases of available fonts. In addition, *xfd* can be used to display the full character set of a particular font, and *xfontsel* can be used interactively to preview and select a font for use in another window.

The Font Displayer: xfd

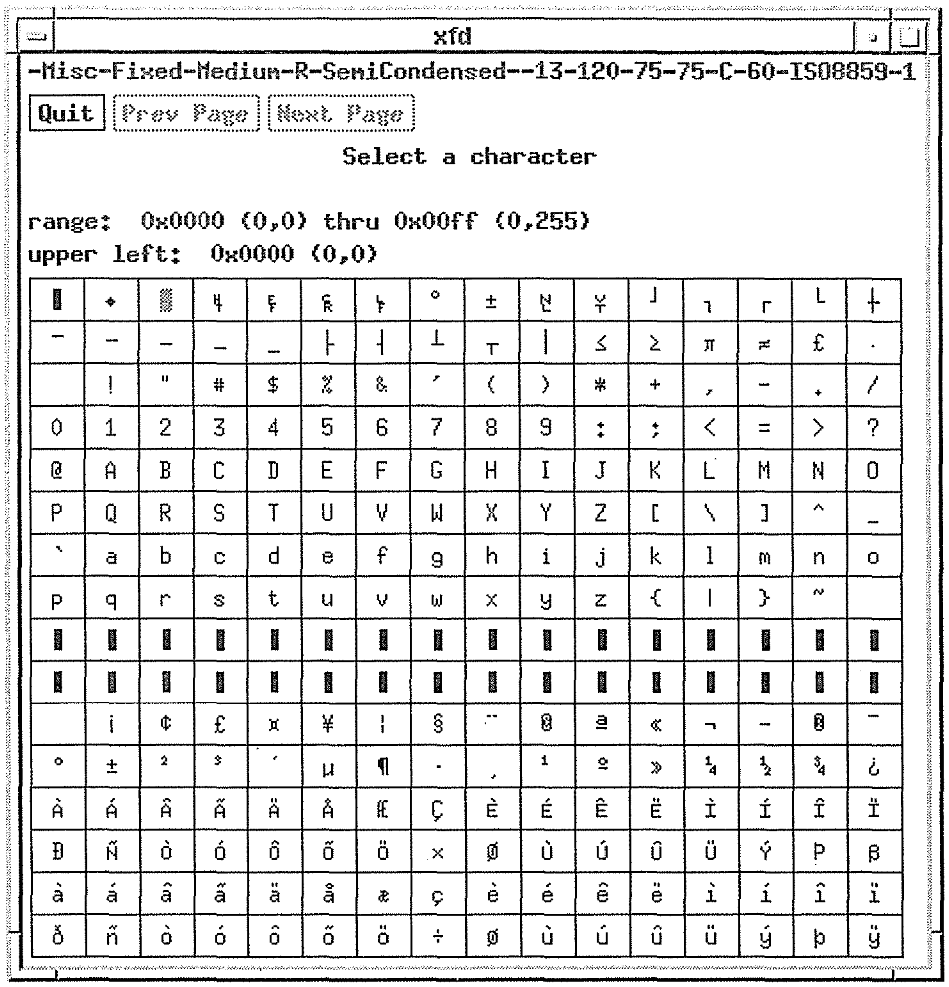
If you're unfamiliar with the general appearance of a particular font, we've included pictures of some representative fonts in Appendix B, *Release 5 Standard Fonts.*

You can also display the characters in a font using the *xfd* (font displayer) client. Note that *xfd* takes the `-fn` option before the font name. For example, to display the default system font, a 6 × 13 pixel fixed-width font known by the alias *fixed*, enter:

```
% xfd -fn fixed &
```

The *xfd* window will display the specified font as shown in Figure 6-7.

*If you don't have the necessary permissions to edit the *fonts.alias* files, simply create a new file in a directory of your choosing. Then use *xset* to add the directory to the server's font search path and to direct the server to reread all *fonts.dir* and *fonts.alias* files, as described in "Modifying the Font Search Path" earlier in this chapter.



```
-Misc-Fixed-Medium-R-SemiCondensed--13-120-75-75-C-60-ISO8859-1
Select a character

range:  0x0000 (0,0) thru 0x00ff (0,255)
upper left:  0x0000 (0,0)
```

▮	◆	▒	␉	␌	␍	␊	°	±	␤	␋	┘	┐	┌	└	┼
⎺	⎻	─	⎼	⎽	├	┤	┴	┬	│	≤	≥	π	≠	£	·
	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮
▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮	▮
	¡	¢	£	¤	¥	¦	§	¨	©	ª	«	¬	-	®	¯
°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Figure 6-7. Fixed font, 6 × 13 pixels

The font name is displayed across the top of the window. (This is the actual font name, which we specified on the command line by the alias *fixed*.) Three command buttons appear in the upper-left corner of the window below the font name. If the font being displayed doesn't fit within a single *xfd* screen, Prev Page and Next Page allow you to scroll through multiple screens. (The horizontal and vertical window dimensions can vary slightly to accommodate different fonts but certain fonts will still require multiple screens.) The Quit button causes the application to exit, though this can also be done by typing q or Q while input is focused on the *xfd* window.

In addition to displaying a font, *xfd* also allows you to display certain information about the individual characters. But before we examine these capabilities, let's take a closer look at the way the characters in a font are identified and how the *xfd* window makes use of this information.

Within a font, each character is considered to be numbered. The *xfd* client displays a font's characters in a grid. By default, the first character of the font appears in the upper-left position; this is character number 0. The two text lines above the grid identify the upper-left character and the range of characters in the window by character numbers both in hexadecimal and in decimal notation (in parentheses following the hex character number).

You can specify a character other than character number 0 to be in the first position in the window using the `-start` option. For example, if you enter this command line:

```
    % xfd -start 15 -fn fixed &
```

the *xfd* window begins with character number 15.

Notice the instruction `Select a character` below the command buttons. To display information about a particular character, click any pointer button within the character's grid square. Statistics about the character's number, width, left bearing, right bearing, ascent, and descent are displayed where the line `Select a character` previously appeared.

The *xfd* client is most useful when you have an idea what font you might want to display. If you don't have a particular font in mind or would like to survey the possibilities, the *xfontsel* client allows you to preview a variety of fonts by specifying each component of the font name using a different menu.

Previewing and Selecting Fonts: xfontsel

The *xfontsel* client provides a font previewer window in which you select the font to view using 14 menus corresponding to the 14 components of a font name. By specifying various font name components, you can take a look at a variety of fonts. This is particularly useful if you are trying to pick good display fonts and you don't have a clear idea what type of font would be best. Rather than running several instances of *xfd*, you can dynamically change the font displayed in the *xfontsel* window by changing the font name components. (Despite the flexibility of *xfontsel*, it's certainly not practical to preview *all* of the available fonts. If you have no idea what a particular font family looks like, see the discussion earlier in this chapter, or refer to Appendix B, *Release 5 Standard Fonts*, for pictures of some representative fonts.)*

Once you've displayed the desired font using the menus, you can make the name of that font the PRIMARY text selection by clicking on the window's select button. You can then paste the font name into another window using the pointer: onto a command line, into a resource file, etc. Making a font name the PRIMARY selection also enables you to choose that font from the *xterm* VT Fonts menu. (Selecting text and using *xterm* menus are described in Chapter 5, *The xterm Terminal Emulator*.)

*To our minds, the major drawback of *xfontsel* is that it shows you only the first font that matches a given wildcarded font name. A far better interface would list all of the matching fonts so that you could compare and choose the one that best suited your needs. There is no way in the standard X distribution to display the appearance of a group of fonts. To produce the figures in this book, we had to write such a program, which we called *xshowfonts*. The program has since been posted to *comp.sources.x*, and a listing appears in Appendix B, *Release 5 Standard Fonts*.

Previewing Fonts with the xfontsel Menus

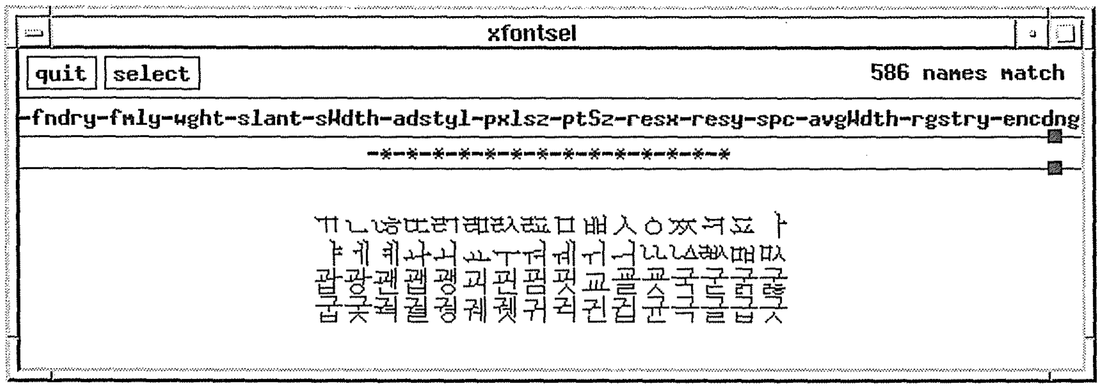
To run *xfontsel*, enter this command in an *xterm* window:

```
% xfontsel &
```

If your system is using the standard Release 5 fonts, the *xfontsel* window initially displays a 12 point, character cell Hangul font (the Korean alphabet) from the *misc* font directory, the complete name of which is:

```
-daewoo-gothic-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0
```

This is the first font in the default font search path. (The foundry "daewoo" comes first alphabetically in the *misc* directory, the first directory in the default font path.) Figure 6-8 shows the initial *xfontsel* window.



```
586 names match
-fndry-fmly-wght-slant-sWdth-adstyl-pxlsz-ptSz-resx-resy-spc-avgWdth-rgstry-encdng
-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Figure 6-8. xfontsel window displaying Hangul font

The upper-left corner of the *xfontsel* window features two command buttons: quit and select. As we've explained, clicking on select with the first pointer button makes the font displayed in the window the PRIMARY text selection; obviously, quit causes the application to exit.

Below the command buttons is, in effect, a generic font name or font name template. It is divided into 14 fields corresponding to the 14 parts of a standard font name. Each field is an abbreviation for one part of a font name. Take a look again at the sample font name in Figure 6-1 to refresh your memory as to the components. Each of the fields in the *xfontsel* window is actually the handle to a menu which lets you specify this part of the font name.

To get a clearer idea of how this works, move the pointer onto the generic font name—specifically onto the first field, fndry. (This is an abbreviation for the first part of a font name, the foundry.) When you place the pointer on fndry, the field title should be highlighted by a box. You can then display a menu of foundry names by pressing and holding down the first pointer button, as in Figure 6-9.

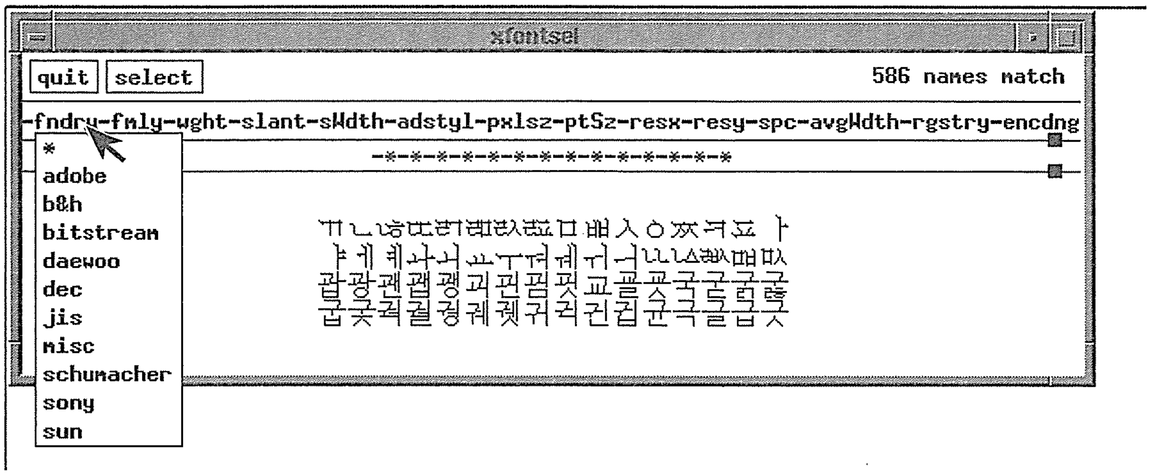


Figure 6-9. xfontsel window with foundry menu displayed

Notice that the first choice is the asterisk (*) wildcard character. This is the first choice on all of the menus, allowing you to include wildcards in the font name you specify rather than explicitly selecting something from all 14 menus.

To specify a font name component (i.e., make a selection from the menu), first display the menu by pressing and holding down the first pointer button. Then move the pointer down the menu. As the pointer rests on each menu item, it is highlighted by reverse video. To select a highlighted menu item, release the first pointer button.

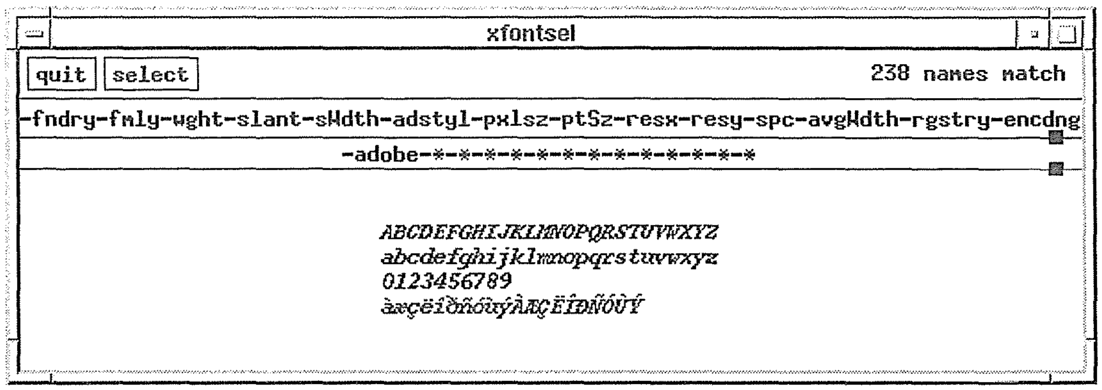
The line below the font name menus represents the actual font name. When you first run *xfontsel*, all of these fields contain wildcard characters because no menu selections have been made. The number of fonts matched by the font name is displayed in the upper-right corner of the window. The number of fonts initially matched depends on the number of fonts with this naming convention available on your system. In this example, 586 fonts match. (Since this line of wildcards can match *any* 14-part font name, the server chooses the first font in the font path that reflects this naming convention.)

When you select a font name component from one of the 14 menus, the component appears in the actual font name, and the *xfontsel* window displays the first font that matches this name. For example, say we select adobe from the fndry menu, the *xfontsel* window would look like Figure 6-10. The font name is now:

```
-adobe-*-*-*-*-*-*-*-*-*-*-*-*-*
```

and the window displays the first font in the font path that matches this wildcarded name. In this case, the first font to match is a 12-point bold Oblique Courier font, which is stored in the file *courBO12.pcf* and has the actual font name:

```
-adobe-courier-bold-o-normal--12-120-75-75-m-70-iso8859-1
```



xfontsel

```
238 names match
-fndry-fmly-wght-slant-sWdth-adstyl-pxlsz-ptSz-resx-resy-spc-avgWdth-rgstry-encdng
-adobe-*-*-*-*-*-*-*-*-*-*-*-*-*
```

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789
```

Figure 6-10. xfontsel after choosing adobe from the foundry menu

Once you make a selection from one menu, the number of possible fonts matched by the name changes. (Notice the line `238 fonts match` in the upper-right corner of the window.) Choosing one font name component also eliminates certain choices on other menus. For example, after you select Adobe as the foundry, the possible choices for font family (the second menu, fmly) are narrowed from 17 to 5 (not counting the asterisk). Again display the fmly menu using the first pointer button. The available choices for font family appear in a regular typeface; the items that are unavailable (i.e., cannot be selected) appear in a lighter typeface. Families such as Clean, Lucida, and Charter are in a lighter typeface because none of the standard X fonts provided by Adobe is from these families. Adobe fonts in the standard X distribution are limited to the five families Courier, Helvetica, New Century Schoolbook, Symbol, and Times; these are the items available on the fmly menu.

In order to display a particular font, you'll probably have to make selections from several of the menus. As described earlier in the section "Font Name Wildcarding," we suggest you explicitly select at least these parts of the font name:

- Font family
- Weight
- Slant
- Point size

Thus, you would make selections from the fmly, wght, slant, and ptSz menus.

You can also use the `-pattern` option with a wildcarded font name to start out with a more limited range of options. For example, if you typed:

```
    % xfontsel -pattern '*courier-bold-o-*'
```

you'd start out with the pattern you specified in the filename template part of the *xfontsel* display. You could then simply select from the ptSz menu to compare the various point sizes of Courier bold oblique until you found the one you wanted.

Note that if the pattern you specify to *xfontsel* matches more than one font, the one that is displayed (the first match found) is the one that the server will use. This is in contrast to *xlsfonts*, which sorts the font names. You can always rely on *xfontsel* to show you the actual font that will be chosen, given any wildcard specification.

As of Release 5, you can use *xfontsel* to preview (and select) a scaled version of any standard font (both bitmap and outline fonts). (As you might remember, though outline fonts are *intended* to be scaled, X supports scaling of bitmap fonts as well.) This capacity of X to scale fonts is reflected in a new ptSz menu. If you display the menu, you'll notice that it offers a wider range of sizes than the actual font files do. To view any of the standard fonts scaled to one of these new sizes:

1. Run *xfontsel* without the `-fn` option.
2. Narrow your selections using the menus, choosing 0 for both pixel size and average width.
3. The point size menu will offer several sizes not generally available in the standard font files. Choose the point size to which you want the font scaled.

If you like the font, you can select it from the *xfontsel* window for use elsewhere, as described in the next section.

Selecting a Font Name

Once you make selections from the menus to compose the name of the font you want, the corresponding font is displayed in the *xfontsel* window. Then you can select that font name by clicking on the select command button with the first pointer button. The font name becomes the PRIMARY text selection and thus can be pasted in another window using the second pointer button, as described in Chapter 5, *The xterm Terminal Emulator*.

You might paste the font name on a client command line in an *xterm* window in order to specify it as the client's display font. (See Chapter 10, *Command-line Options*.) You might paste it into a resource file such as *.Xresources* to specify it as the default font for a client or some feature of a client (such as a menu). (See Chapter 11, *Setting Resources*, for more information.)

Less obviously, once a font name is made the PRIMARY text selection, it can be toggled as the *xterm* display font using the Selection item of the *xterm* VT Fonts menu. The Selection menu item can only be chosen from the VT Fonts menu when there is a PRIMARY text selection. (Otherwise, the menu item appears in a lighter typeface, indicating that it is not available.) If the PRIMARY text selection is a valid font name (as it is when you've pressed the select button in the *xfontsel* window), the *xterm* window displays in that font. (In cases where the PRIMARY selection is not a valid font name, the *xterm* display font does not change.)

By default, *xfontsel* displays the alphabetic characters (and the digits 0 through 9, if the character set includes them). You can specify alternative sample text using the `-sample` option. For more information about this and other options, see the *xfontsel* reference page in Part Three of this guide.

Changing Fonts in xterm Windows

xterm includes a VT Fonts menu that allows you to change fonts on the fly. We discussed most of the menu entries in Chapter 5. However, two of the many items require a greater understanding of font naming than we'd covered by that point. So we've saved them until now.

The Great Escape

Though it is by no means obvious, *xterm* allows you to change the display font by sending an escape sequence, along with the new font name, to the terminal window. Once you change the font in this way, the Escape Sequence item on the *xterm* VT Fonts menu becomes available and choosing it toggles the font you first specified with the escape sequence. (In effect, whatever font you specify using the escape sequence is stored in memory as the menu's Escape Sequence font selection.)

You send an escape sequence to the terminal window by using the UNIX *echo*(1) command. The escape sequence to change the *xterm* display font comprises these keystrokes:

```
Esc ] 50 ; fontname Control-G
```

To clarify, these keystrokes are: the Escape key, the right bracket (]), the number 50, a semicolon (;), a *fontname*, and the Control-G key combination. We've shown the keystrokes with spaces between them for readability, but when you type the sequence on the command line, there should be no spaces. Note also that to supply this sequence as an argument to *echo*, you must enclose it in quotes:

```
% echo "Esc]50;fontnameControl-G"
```

These are the literal keys you type. However, be aware 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, the command line will actually look like this:

```
% echo "^[]50;fontname^G"
```

Pressing the Escape key generates the ^[symbol; typing the Control-G key combination generates ^G. You can use a full fontname, an alias, or a wildcarded font specification as the font name. You should be aware that if the wildcarded specification matches more than one font, you will get the first font in the search path that matches. For example:

```
% echo "^[]50;*courier*^G"
```

will get you a 10-point courier bold oblique. The advantage of being able to change the display font with an escape sequence is that it allows you to add another font to your choices on the fly.*

Changing the fonts associated with the Unreadable, Tiny, Small, Medium, Large, and Huge menu items is a more laborious process. It involves specifying other fonts in a resource file, making those resources available to the server, and then running another *xterm* process. (See Chapter 11, *Setting Resources*, for more information.) However, you can change the font specified by the Escape Sequence menu item as often as you want during the current *xterm* process, simply by typing the escape sequence described previously.

Now that we've looked at the mechanics of the escape sequence, let's consider its practical use. Say you want to run a program in an *xterm* window and you want to be able to read the output easily, but you would like the window to be moderately small. You discover that toggling the Medium font, the `7x13` font by default, makes the window a good size, but the typeface is too light to be read easily. (We presume you are using the default menu fonts and have not customized them using a resource file.) You could dynamically change the display font to a bold font of the same size by entering the following command line:

```
% echo "Esc]50;7x13boldControl-G"
```

The *xterm* font becomes the desired `7x13bold`, a good choice; in addition, the Escape Sequence item of the VT Fonts menu becomes available for selection. This menu item allows you to toggle the `7x13bold` font at any time during the *xterm* process. Thus, you could switch back to any of the other fonts available on the menu (Small, Large, etc.) and then use Escape Sequence to again select `7x13bold`.

This font will remain the Escape Sequence font for the duration of the *xterm* process, unless you again change the display font with an escape sequence. If you enter another font name using the escape sequence described above, the window will display in that new font and the Escape Sequence menu item will toggle it.

The Selection Menu Item

The Selection menu item allows you to toggle a font whose name you've previously "selected." For example, you could select the font name with the pointer from *xlsfonts* output, using the "cut-and-paste" techniques described in Chapter 5, *The xterm Terminal Emulator*. It is far more likely, though, that you would use this menu item after selecting a font with *xfontsel*. This menu item was clearly designed with *xfontsel* in mind. (If no text is currently selected, the Selection menu item appears in a lighter typeface, indicating that it is unavailable.)

*Specifying a font with an escape sequence affects only the current *xterm* window and enables only that window's Escape Sequence menu selection.

The main limitation of this menu item is that it uses the *last text selected* as the font name, regardless of what that text is. If you select a font name, that name is only available through Selection until you use the pointer to select other text. Since cutting and pasting text is one of the most useful features of *xterm*, you will probably be making frequent selections. If the last selected text was not a valid font name, toggling Selection will not change the display font, and a beep will inform you that the toggle failed.

Using the Font Server

The font server (*fs*) program acts as intermediary between your X server and the fonts resident on the system where the font server is running. If the font server is running on one or more remote machines, you can add those servers to your font path on the local host and then access the remote fonts.* Once a font server is added to your path, you can request a font in the normal manner (e.g., on the command line, in a resource file, etc.) and both the local and remote directories will be searched. The access is invisible.

You can also add a font server running on the local system to your font path, though the advantages of this are more subtle. Primarily, having the local font server in your path enables you to use certain "Font Server Clients" (described later in this chapter) with local fonts.

The operation of the font server program is controlled by a special configuration file called *config*, which is commonly located in the directory */usr/lib/X11/fs*. The sample configuration file provided in the MIT distribution should be sufficient in many environments.

In the following section, we'll take a quick look at the sample configuration file. Then we'll see how to run the font server program and how to add servers to the font path. (Keep in mind, however, that the system administrator for the machine in question will generally be the one to run *fs*.) Finally, we'll consider some of the clients you can run to find out information about the font server and the available fonts.

Be aware that environment- and vendor-specific issues may necessitate customizing the *config* file. Volume Eight, *X Window System Administrator's Guide*, provides instructions on customizing the *config* file and explains how to run the font server in a variety of environments.

*Technically, if the font server is not running on a host whose fonts you want to use, you can run it yourself—presuming you have access. We'll show you a rudimentary way to do this; however, in most cases, running *fs* will be up to the system administrator. See Volume Eight, *X Window System Administrator's Guide*, for more information.

The Font Server config File

The font server *config* file has a fairly simple syntax, though some of the variables it sets may be unfamiliar. (They deal with functionality that is generally the province of the system administrator.) Here's the sample file provided with the standard distribution:

```
# font server configuration file
# $XConsortium: config.cpp,v 1.7 91/08/22 11:39:59 rws Exp $

clone-self = on
use-syslog = off
catalogue =
/usr/lib/X11/fonts/misc/,/usr/lib/X11/fonts/Speedo/,/usr/lib/X11/fonts/75
dpi/,/usr/lib/X11/fonts/100dpi/
error-file = /usr/lib/X11/fs/fs-errors
# in decipoints
default-point-size = 120
default-resolutions = 75,75,100,100
```

This file sets six variables:

`clone-self`	The default value of on specifies that the font server should spawn another copy of itself to handle more than 10 client connections (the default maximum).
`use-syslog`	The default value of `off` specifies that error messages will not be logged (using *syslog*).
`catalogue`	A list of the font directories available from the server. (By default, this is the standard font path.)
`error-file`	An error log file (an alternative to *syslog*).
`default-point-size`	In decipoints (or tenths of a point); thus, the default is 12 points.
`default-resolutions`	Typical horizontal and vertical resolutions.

The defaults the file provides should be reasonable for most environments. You would need to change the `catalogue` to match an alternative font path.

Running the Font Server

There's a reasonable chance that the font server will already be running on the hosts you want to access. (The system administrator may keep the font server running all the time by placing it in a system startup file, such as */etc/rc.local*.) You can verify that the font server is running either by asking the system administrator or by logging in to the system in question and searching for the relevant process number:

```
% ps -aux | grep fs
```

(Depending on the system, you may need to use an analogous command.) If the font server is running, you'll see output containing a line similar to the one in bold:

```
             .
             .
             .
root        103  0.0  0.0   40    0 ?  S     Sep 30  0:02  (nfsd)
root        104  0.0  0.0   40    0 ?  S     Sep 30  0:02  (nfsd)
root        366  0.0  0.0 4496    0 p3 IW    Oct  3  4:40 /usr/bin/X11/fs
val        1069  0.0  3.0   40  224 p3 S     19:20   0:00 grep fs
```

which corresponds to the *fs* process. If the font server is running, you can skip ahead to the next section, "Adding a Server to the Font Path."

If *fs* is not running on a host you want to use as a font server, ask the system administrator to start it or to give you permission to start it from the command line. If you need to start the font server yourself, log in to the machine you want and enter:

```
% fs &
```

An error message such as:

```
Error: Can't open error file "/usr/lib/X11/fs/fs-errors"
```

means that the font server probably doesn't have write permission for the error file. Any errors will be sent to the controlling *xterm* or the console. You can specify an another file with the `error-file` keyword in the font server *config* file; or ask the system administrator to add write permission to *fs-errors*.

If for some reason you can't run *fs* or if you want to configure the font server in a particular way, speak to the system administrator. Volume Eight, *X Window System Administrator's Guide*, describes several possible font server configurations.

Adding a Server to the Font Path

When the *fs* (font server) program is running on a remote host, you can identify the host as a font server to access from your local machine. To specify that a remote font server should be searched along with the current (local) font path, you use the *xset* client.

The format of a font server name is:

```
transport/hostname:port
```

`transport` refers to the protocol used by the font server. For UNIX environments, the protocol is `tcp`. The default *`port`* is `7000`. Here is a typical font server name:

```
tcp/ruby:7000
```

Say you're working on a machine called *harry*, but you want to access fonts available on another machine called *ruby*. Assuming that the *fs* program is running on *ruby* (a reasonable assumption), you simply have to add the the font server *ruby* to your font path on *harry*. (We'll use a "harry%" prompt to make things clearer.)

```
harry% xset fp+ tcp/ruby:7000
```

If you get no error message, query the font path setting using *xset* to verify that the server *ruby* has been added.

```
harry% xset q
        .
        .
        .
Font Path:
 /usr/lib/X11/fonts/misc/,/usr/lib/X11/fonts/Speedo/,/usr/lib/X11/fonts/75dpi/,
/usr/lib/X11/fonts/100dpi/,tcp/ruby:7000
```

If *fs* is not running on *ruby* or if you've supplied an incorrect server name, when you try to add a font server to the path you will get the error message:

```
X Error of failed request:  BadValue (integer parameter out of range
for operation)
  Major opcode of failed request:  51 (X_SetFontPath)
  Value in failed request:  0x5
  Serial number of failed request:  5
  Current serial number in output stream:  8
```

If *fs* isn't running on *ruby*, you can speak to the system administrator or try to run it on the command line yourself.

Note that you can also add an instance of the font server running on the local machine to your path. The following command prepends the font server running on *harry* to your font path on *harry*:

```
harry% xset +fp tcp/harry:7000
```

We can verify this by checking the path:

```
harry% xset q
        .
        .
        .
Font Path:
 tcp/harry:7000,/usr/lib/X11/fonts/misc/,/usr/lib/X11/fonts/Speedo/,\
/usr/lib/X11/fonts/75dpi/,/usr/lib/X11/fonts/100dpi/,tcp/ruby:7000
```

If you have access to a local font server, you can use the following clients with local fonts.

Font Server Clients

Once you've added a font server to your path, you can specify a font from that host on the command line or in a resource file. You can also use a few clients that are intended to be run with the font server.

fslsfonts is analogous to *lsfonts* in that it list the names of all fonts available from a specified font server, as in the following example:

```
harry% fslsfonts -server tcp/ruby:7000
```

Since this output is sizeable, it's a good idea to pipe it to a paging program like *more*:

```
harry% fslsfonts -server tcp/ruby:7000 | more
```

You can also limit your search to a particular font (an alias is acceptable) or a group of fonts (a wildcarded name in quotes) by using the additional command-line option, `-fn`:

```
harry% fslsfonts -server tcp/ruby:7000 -fn fixed
fixed
```

Notice that the program echoes back the alias "fixed," indicating that the font is available from the server *ruby*.

If you get an error such as:

```
fslsfonts: pattern "huh" unmatched
```

you may have specified an invalid font name, or the font server *config* file may have an error in one of the path names.

The *fsinfo* client simply verifies that a server is running and lists alternative servers (in this case, none):

```
harry% fsinfo -server tcp/ruby:7000
name of server: tcp/ruby:7000
version number: 1
vendor string:  MIT X Consortium
vendor release number:  5000
maximum request size:   16384 longwords (65536 bytes)
number of catalogues:   1
        all
Number of alternate servers: 0
number of extensions:   0
```

The *showfont* client provides some rather arcane information about a font (much of which is provided without a font server by *xfd*). *showfont* also provides an ASCII representation of each character in a font. (You can convert a character into a bitmap image using the *atobm* program. See Chapter 7, *Graphics Utilities*, for instructions.)

To gather data about a font from a server, you'd use a command line similar to:

```
harry% showfont -server tcp/ruby:7000 -fn fixed > fixed.info
```

In our example, we've redirected the information to a file. You might instead pipe it to a paging program, as in the *fslsfonts* example.

In Chapter 7, *Graphics Utilities*, we grab the Gumby character from the cursor font and convert it into a bitmap image. If you're working on *harry* and the local font server is in your path, you can save the cursor font to a file by entering:

```
% showfont -server tcp/harry:7000 cursor > /tmp/cursor.array
```

Notice that we've used the alias for the cursor font.

See Chapter 7, *Graphics Utilities*, for instructions on converting a font character to a bitmap using *atobm*.

For more information on *fslsfonts*, *fsinfo*, and *showfont*, see the client reference pages in Part Three of this guide.

7

Graphics Utilities

This chapter describes how to use the major graphics clients included with X, notably the bitmap *editor.*

In This Chapter:

☞

7 Graphics Utilities

A *bitmap* is an image composed of individual pixels, or picture elements (you might think of them as dots), each of which is white, black, or, in the case of color displays, a color. Most of the images on an X display are actually bitmaps. For example, the various cursor symbols representing the pointer on the screen, such as the arrow, crosshair, and I-beam, are bitmap images (in this case, incorporated into the cursor font).

The standard release of X includes four utilities to help you create bitmap images: *bitmap*, *xmag*, *bmtoa*, and *atobm*. The most powerful and useful of these clients is *bitmap*, a program that lets you create and edit bitmap files. This chapter provides detailed instructions for using the *bitmap* client. You can use *bitmap* to create images to use as backgrounds, icons, and pointer symbols.

In a sense, the *xmag* client is a desk accessory for graphics programs. This client is used to magnify a portion of the screen, to assist you in creating images with a graphics editor, such as *bitmap*, or in capturing screen images with the *xwd* window dump utility, described in Chapter 8, *Other Clients*. As of Release 5, *xmag* is designed to work in concert with *bitmap*. You can copy and paste bitmap images between these clients. This chapter provides all the information you need to get started with *xmag*.

The *bmtoa* and *atobm* clients are programs that convert bitmaps to arrays (of ASCII characters) and arrays to bitmaps. They are used to facilitate printing and file manipulation and can help you convert a font character to a bitmap, as we'll demonstrate.

In addition to these standard clients, the user-contributed part of the X distribution includes a very useful library of graphic utilities: the Portable Bitmap Toolkit, developed by Jef Poskanzer. The PBM Toolkit is made up of dozens of programs that convert graphic images to and from portable formats. Some of the conversions you can perform are described later in this chapter.

Overview of the bitmap Editor

The *bitmap* program allows you to create and edit small bitmaps. At this point in X Window System development, *bitmap* is primarily a tool for application developers. However, several clients allow you to specify a bitmap as an icon, cursor symbol, or background pattern. Thus, you can create a bitmap image using *bitmap*, save it in a file, and specify that filename on the command line.* For example, *xsetroot* (described in Chapter 14, *Setup Clients*) allows you to specify a bitmap that will be used as the background pattern for the root window or as the root window pointer.

To bring up an empty *bitmap* window in which to edit, type:

```
    % bitmap &
```

Note that no filename argument is necessary. (The Release 4 *bitmap* requires one.) The default *bitmap* window is shown in Figure 7-1.

The window that *bitmap* creates has three sections: a menu bar, a column of editing command buttons, and the actual editing area.

Menu Bar

The menu bar across the top of the application window provides two menus, File and Edit. You display a menu by placing the pointer on the appropriate command button and pressing and holding down any pointer button. More about the available menu options later. The menu bar also displays the name of the bitmap file. When you run a new instance of the program, this name will be *none*.

Editing Command Buttons

Below the menu bar, on the left side of the window, is a list of editing commands in buttons. Place the pointer on a command button and click the first pointer button to invoke the command. These commands help you draw the bitmap image. The section "Bitmap Command Buttons" later in this chapter provides an overview of these functions.

*There are many bitmaps included in the X distribution. These can generally be found in the directory */usr/include/X11/bitmaps*. Samples are shown in Appendix C, *Standard Bitmaps*.

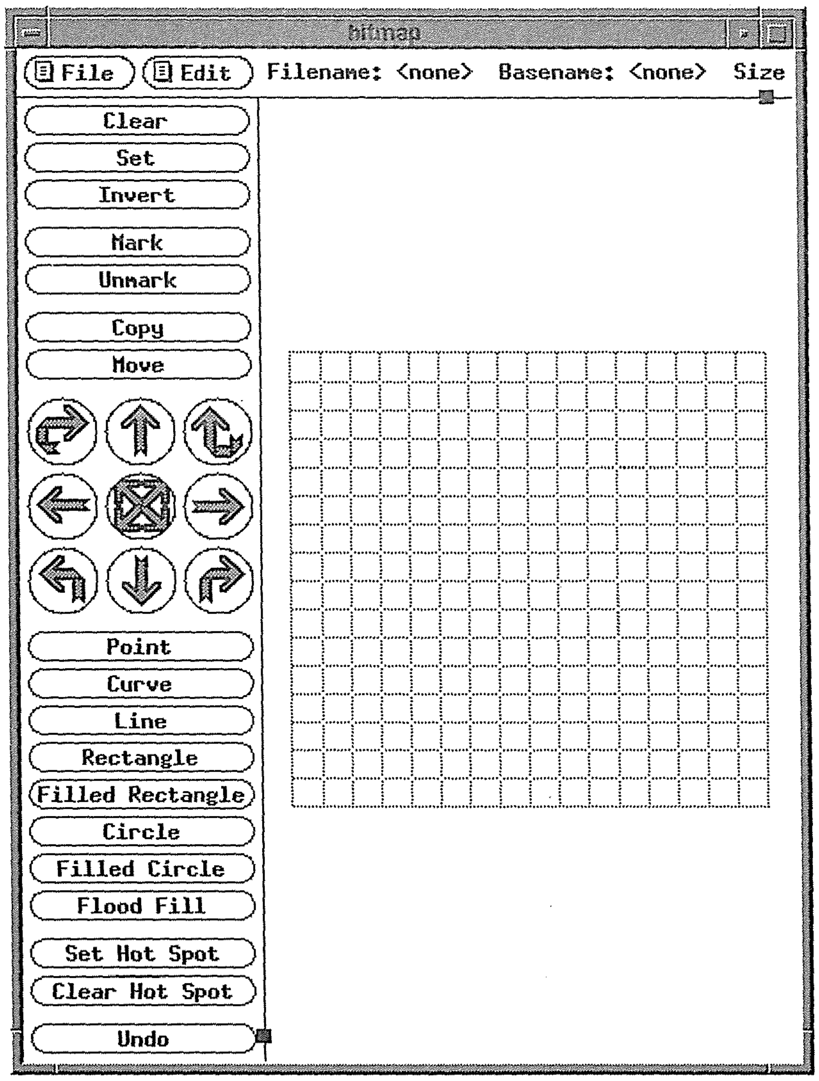


Figure 7-1. Release 5 Bitmap window

Editing Area

To the right of the command buttons is a grid in which you create/edit the bitmap. Each square in the grid represents a single pixel on the screen. The default size of the grid is 16 × 16 squares, representing an area of the screen 16 × 16 pixels—a fairly tiny spot. The grid affords a close-up look at this area and allows you to edit an image that would otherwise be too small to work with.

Once you begin playing with *bitmap*, you may find the 16 × 16 space too restricting. You can specify another grid size using the `-size` command-line option, which has the syntax:

```
-size widthxheight
```

The following command line opens a *bitmap* window with a 32 × 32 grid, a more workable space:

```
% bitmap -size 32x32 &
```

The `-size` option is new in Release 5. In prior releases, you could supply the dimensions simply as a command-line argument, without a preceding option.

Figure 7-2 shows a 40 × 40 grid with a bitmap we created of Gumby. We think it makes a fun root window pattern. (See the discussion of *xsetroot* in Chapter 14, *Setup Clients*, for instructions on specifying a bitmap as your root window pattern.)

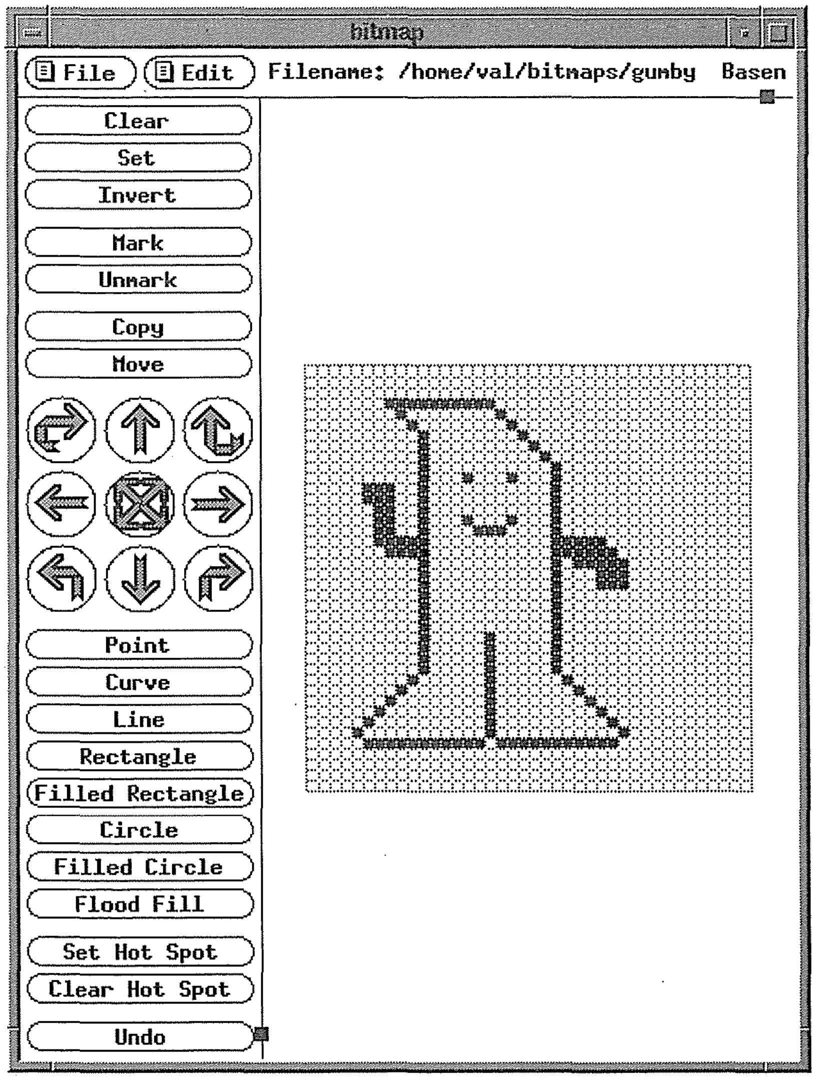


Figure 7-2. Gumby bitmap

The standard cursor font also contains a Gumby character. (You can specify the Gumby cursor as the *xterm* window pointer, as described in Chapter 11, *Setting Resources*, or as the root window pointer using the *xsetroot* client, as described in Chapter 14, *Setup Clients*.) Later in this chapter, we'll show you how to convert the Gumby character of the cursor font to a bitmap file using the *atobm* client.

Image Size Versus Window Size

Keep in mind that there is an interaction between the size of the bitmap image being edited and the size of the *bitmap* window. By default, each cell in the *bitmap* editing area is 13 pixels square. If you specify grid dimensions of 40 × 40, the editing area alone will be 520 pixels square. Specifying a very large editing area may even result in an application window larger than the screen. (Since *bitmap* does not provide a scrollbar, a large window makes it very difficult to edit!)

You can change the size of the editing area (not the number of squares) by: specifying an explicit size for the overall application using the `-geometry` option; or resizing the application window using the window manager. In either case, *bitmap* will automatically adjust the size of each square in the grid to fit the overall window size. (If the bitmap image is very large and *bitmap* window cannot incorporate it, the image will appear truncated. If you resize the window to be larger, more of the image should be revealed.)

A mild caution about using the `-geometry` option: when you specify a size for the overall application window, it may be difficult to estimate what size the editing squares will be. So if you use `-geometry`, there's a reasonable chance you'll have to resize anyway.

You can control the size of the editing grid even more precisely by using the `-sw` and `-sh` command-line options, which set the width and height of each square in pixels. We find that we need at least 10 × 10 pixel grid squares to edit comfortably, but this is clearly an individual preference.

As you're probably realizing, the number of factors that can determine the size of the application window and the editing grid make specifying the size on the command line rather complicated. Your best bet may be to confine yourself to the `-size` option for a while and make any necessary adjustments using the window manager. (You may never need to use `-geometry`, `-sw`, or `-sh`.)

Once you run *bitmap*, you can adjust either the size of the squares in the grid, or the actual number of squares. You change the size of squares in the grid by resizing the *bitmap* window using the window manager. For example, if the cells in the grid aren't large enough for comfortable editing, make the window larger. Each square on the grid will be enlarged proportionally. Alternatively, you might have a grid with large dimensions and make the window smaller so it takes up less space. The squares will be made smaller proportionally. (Note, however, when the squares are made very small, the grid lines are suppressed. Though editing is still possible in such circumstances, it is difficult.)

You can change the number of squares in the editing grid using the Resize option on the File menu. (See "The File Menu" later in this chapter.) Of course, if you want, you can also change the dimensions of each square using the window manager!

You can open an existing bitmap file to edit by supplying the filename on the *bitmap* command line. Keep in mind that if no size is specified and the image you display is larger than 16 × 16, *bitmap* will try to fit the image into an editing area the same overall size as the default. However, to accomplish this, each pixel in the image will be scaled down and the grid lines will be suppressed. In this case, you can still use all the editing commands, but chances are the image will be too small to work with precisely. If the image is large enough, one pixel in the image may actually be represented by a single pixel in the editing area! Some images may even be too large to fit under this circumstance, in which case the image will appear truncated. If you resize to make the window larger, more of the image should be revealed. As a general rule, if you know the dimensions of the image in pixels, it's a good idea to give them with `-size`.

What's New in Release 5

The Release 5 *bitmap* is actually much more powerful and versatile than the Release 4 version, but is unfortunately somewhat less intuitive. If you're using *bitmap* for the first time, you can skip this section altogether. If you're trying to switch from R4 to R5, you might want to at least glance through this section.

As mentioned previously, a filename argument is no longer required on the command line. The `-size` option, which allows you to specify the dimensions of the editing grid, is also new in Release 5. See the section "Editing Area" earlier in this chapter for some syntax examples and the *bitmap* reference page in Part Three of this guide for a list of valid options.

Release 5 sees the addition of a menu bar, which provides two menus, File and Edit, each offering many useful new functions. There are also keyboard shortcuts for all menu commands (as well as for some of the command buttons). Among the new functions provided on the menus are those that allow you to:

- Dynamically load another bitmap file into the editing window.
- Insert another bitmap image into the current bitmap file.
- Save one image and start editing a new one.
- Change the current filename.
- Change the dimensions of the grid.
- Copy/cut and paste between *bitmap* windows (or between *bitmap* and *xmag*).

In addition to the new functionality provided by the menus, there are additional editing command buttons (on the left side of the window). You can now draw a Rectangle or Filled Rectangle. Don't miss the Undo command, a long overdue improvement. Also helpful (if somewhat less than intuitive) are "arrow" command buttons (halfway down in the window) to flip or move the image (or part of it).

There is perhaps one obvious negative change to *bitmap*: by default the Release 4 *bitmap* shows you an actual size picture of the bitmap—in a tiny box below the command buttons (also an inverted version of the same bitmap). Each time the grid changes, the same change occurs in the actual size bitmap and its inverse. In Release 5, you must select the Image item of the Edit menu to bring up these two tiny images, which then appear in a small window on top of the *bitmap* window, above and to the left of the editing grid.

Note also that the command buttons to Clear Area, Set Area, and Invert Area have been removed. You can perform the equivalent of Clear Area by marking an area (Mark command button) and using the Cut command (on the Edit menu). Flood Fill can do in place of Set Area in some cases. There is no replacement for Invert Area.

bitmap Editing Commands

You can create and edit a bitmap using a combination of pointer commands and command buttons (on the left side of the window), and certain File and Edit menu items. The pointer commands work on one grid square at a time, while the relevant Edit menu items and many of the command buttons can work on the entire grid or on a specified area.

Pointer Commands to Draw

If you want, you can draw an image pixel by pixel simply by using the pointer (though the command buttons and menu items can make things easier). When you first open a *bitmap* window, before you select any commands, the following pointer actions are in effect. The same actions also apply whenever you've selected any of the drawing command buttons (Point, Curve, Line, Rectangle, Filled Rectangle, Circle, and Filled Circle).

Note that each pointer button has a different effect on the single square under the pointer.

First button	Changes a grid square to the foreground color and sets the corresponding bitmap bit to 1. (On most displays, the default foreground color is black.)
Second button	Inverts a grid square, changing its color and inverting its bitmap bit.
Third button	Changes a grid square to the background color (often white) and sets the corresponding bitmap bit to 0.

If you click the pointer, you change one square at a time (regardless of which drawing command is in effect). You can hold down a pointer button and drag the pointer to change several squares in a row. Depending on which drawing command is in effect, the shape you can draw in this way is limited. See "Drawing: Point, Curve, Line, Rectangle, Circle."

Pointer actions in the whitespace surrounding the grid affect the outermost row on that side of the grid.

Note that if any of the drawing commands is in effect and you click in the editing area to raise the window, you will change the pixel closest to the pointer.

Bitmap Command Buttons

The command buttons on the left side of the window are intended to make drawing and editing easier. To invoke a *bitmap* editing command, move the pointer to the appropriate command button and click the first pointer button. When you select a command button, the button is inverted (i.e., it displays in reverse video). Most choices are mutually exclusive. For example, you can't draw a Curve and a Line at the same time. Thus, only one of those command buttons can be active at a time.

Most of the commands have keyboard *accelerators*, or shortcuts. To use a shortcut, the pointer must rest inside the editing grid or the surrounding whitespace (not in the menu bar or the command button column).

Note that, as of Release 5, *bitmap* provides an Undo command (and a keyboard accelerator for it, u).

When you start *bitmap*, the Point command is in effect (though there is no indication—the button is not inverted). Point means that you can use the "Pointer Commands to Draw" described previously.

The following sections describe the various command buttons in what we consider to be a logical order, rather than the order in which they appear in the window.

Undo

Last in the *bitmap* window but perhaps first in importance is the Undo command button, which negates the effect of the last action you performed. Typing u while the pointer rests in the editing area has the same effect.

Drawing: Point, Curve, Line, Rectangle, Circle

Though you can draw pixel by pixel by clicking the pointer (as described in "Pointer Commands to Draw" earlier in this chapter), *bitmap* provides several command buttons that can make drawing easier. If you use the first pointer button to draw, the drawing is done in the foreground color. If you use the third pointer button, the drawing is done in the background color. (The second button inverts each square in the area drawn.)

Point	As we've said, Point is in effect when you start *bitmap*. It allows you to draw using the pointer on individual pixels. If you click the pointer, you change one pixel. If you hold down a pointer button and drag the pointer, you change the pixels over which the pointer travels.

Curve Curve (new in R5) is intended to let you draw continuous curved lines—a good idea. Note, however, that drawing the curve you want is not necessarily easy. Curve merely insures that when you hold a pointer button down and drag (as you can with Point, Line, etc.), you'll get a *continuous* line. The way you move the pointer determines the arc (if any). (Curve allows you to draw a straight line as well.) To draw a curve:

1. Click the first pointer button over the Curve command button. The button displays in reverse video.
2. Move the pointer to the editing grid. The pointer becomes the crosshair symbol.
3. Hold a pointer button and drag the pointer to draw a curve. The button you use determines whether the curve will be drawn using the foreground or background color. (See "Pointer Commands to Draw" earlier in this chapter.)

Figure 7-3 shows a curve drawn in the foreground color (by holding the first pointer button and dragging).

Line If you select Line and then hold down and drag the pointer in the grid, *bitmap* will draw a continuous line that is *as straight as possible*. Again, the pointer button you hold determines the color in which the line is drawn. Note that this command works differently than in previous releases in which you chose the beginning and ending points of the line and *bitmap* connected them. To draw a line:

1. Click the first pointer button on the Line command button. The button displays in reverse video.
2. Move the pointer into the editing area. It changes to a cross symbol.
3. Place the cross pointer on the square you want to be one end of the line. (The left end is easier for a righthanded person, right end for someone left-handed.) Press and hold down the first pointer button to draw in the foreground color. A gray shading covers the square the pointer is on.
4. Drag the pointer away from that square. The line you trace with the pointer will be highlighted in gray.
5. When you've traced the line you want (still in gray shading), release the pointer button. The line will be drawn (in this case in the foreground color).

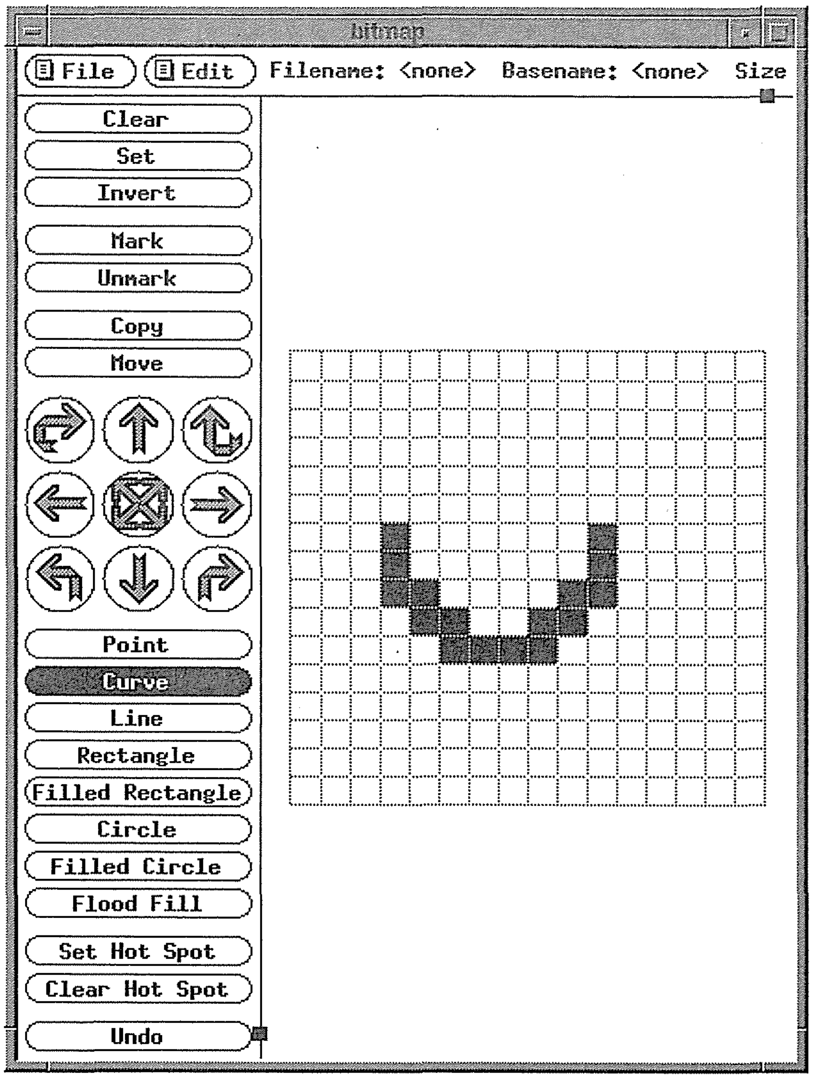


Figure 7-3. Drawing a curve

In Figure 7-4 we've drawn a diagonal line from the bottom left corner to the top right corner of the editing area.

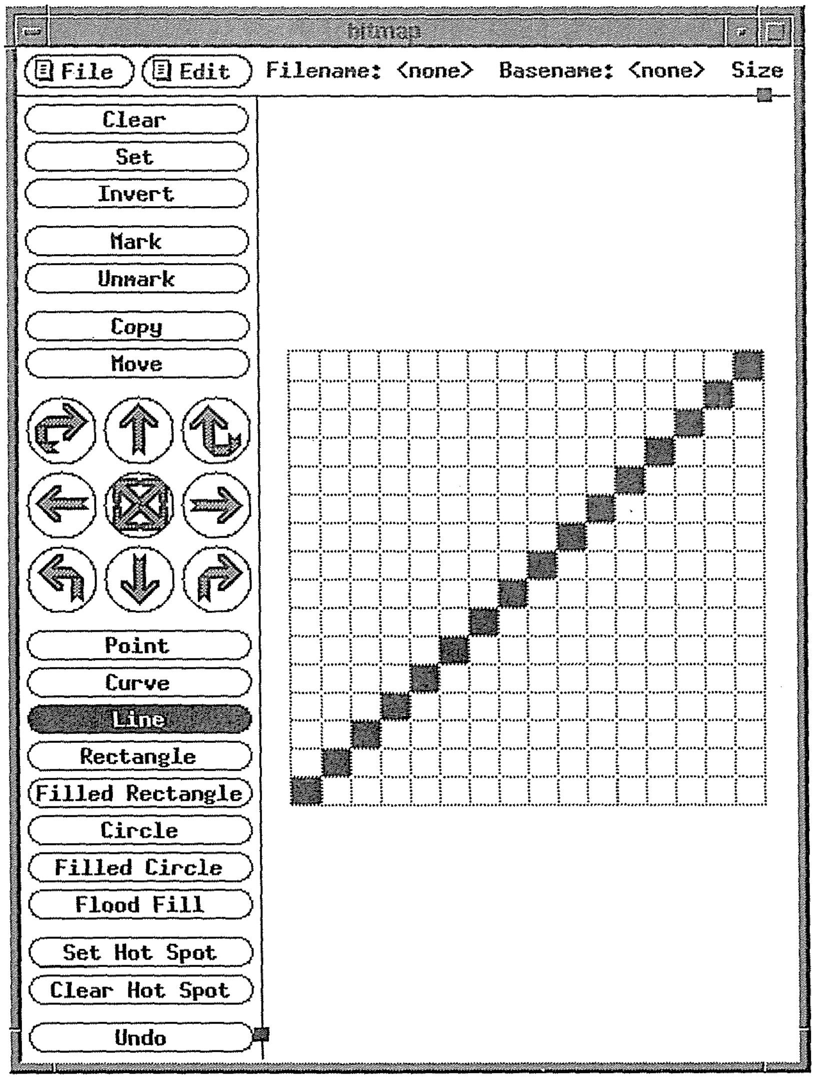


Figure 7-4. Drawing a line

Rectangle and Filled Rectangle Rectangle and Filled Rectangle are new in R5. They allow you to draw an outline of a rectangle and a solid rectangle, respectively. Here's how to draw a rectangle:

1. Click the first pointer button on the Rectangle (or Filled Rectangle) command button. The button displays in reverse video.
2. Move the pointer into the editing area. It changes to a cross symbol.
3. Place the cross pointer on the square you want to be the upper-left corner of the rectangle. (Upper-left is easy for a righthanded person, but you can actually drag from any corner.) Press and hold down any pointer button. A gray shading covers the square the pointer is on.

4. Drag the pointer away from that square. The area you cover with the pointer will be outlined in gray. In our example, we drag the pointer diagonally to the right, highlighting an area 6 squares wide by 4 squares high.

5. When you've specified the outline of the rectangle you want, release the pointer button. The rectangle will be drawn.

In Figure 7-5 we've drawn a plain rectangle.

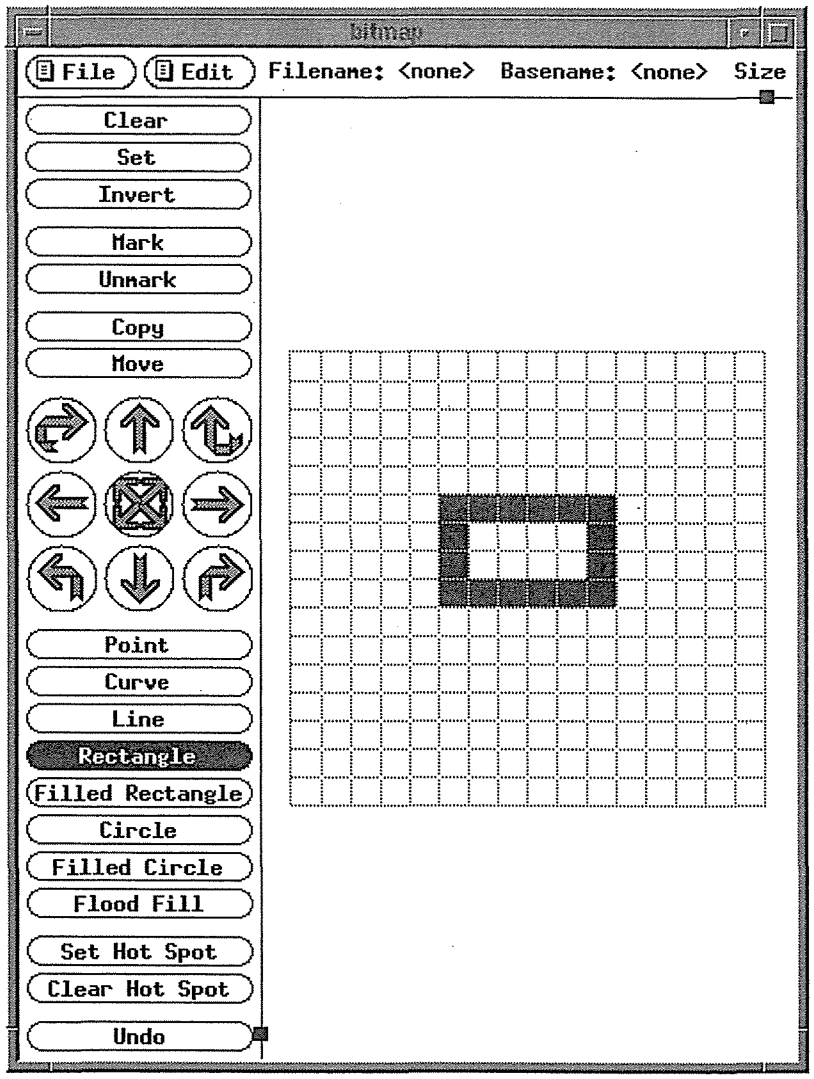


Figure 7-5. Drawing a rectangle

Circle and Filled Circle

Circle and Filled Circle are carried over from Release 4, but (like Line) work differently in Release 5. In R4, you click twice, to mark the center of the circle and a point on the perimeter, and bitmap draws the circle accordingly. To draw a circle in R5:

1. Click the first pointer button on the Circle (or Filled Circle) command button. The button displays in reverse video.
2. Move the pointer to the grid square you want to be the center of the circle.
3. Press and hold the first pointer button (to draw in the foreground color). The selected square is grayed out.
4. Now drag the pointer out toward the perimeter you want. Gray shading tracks the circle as you draw it.
5. When the circle is the size you want, release the pointer button. The circle is drawn.

In Figure 7-6 we've drawn a filled circle.

Filling in a Shape: Flood Fill

Flood Fill changes all squares in a closed shape. Most people probably think of filling a hollow shape with the foreground color. (You can do this by selecting Flood Fill and then clicking on the shape with the first pointer button.) But you can click with the third pointer button to change the squares to the background color. (The second pointer button will invert, but is not particularly useful in this case; if you experiment a bit, you'll see buttons one and three cover all possible situations.) To "fill" an area:

1. Click the first pointer button on the Flood Fill command button. The button displays in reverse video.
2. Move the pointer to the area of the grid you want to fill.
3. Click the first pointer button to change the squares to the foreground color.

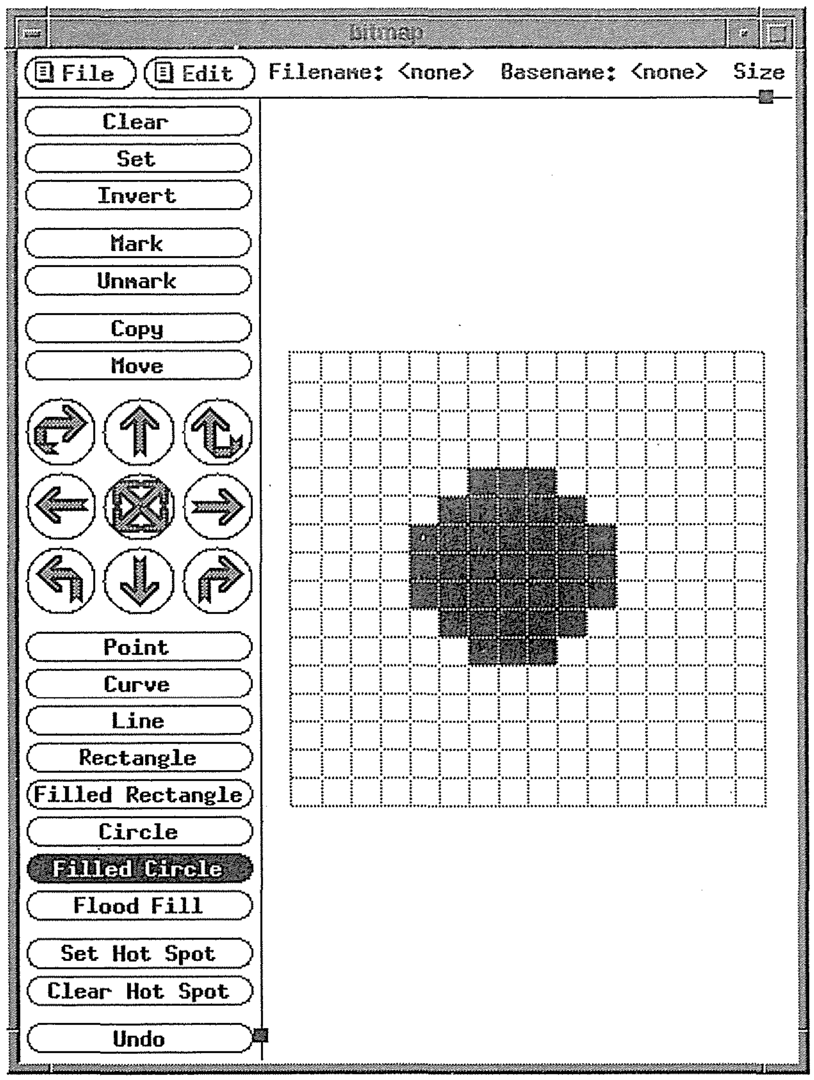


Figure 7-6. Drawing a filled circle

Figure 7-7 shows three concentric circles and then the result of filling the second circle.

Note that you could create the same bitmap just by using the circle drawing commands creatively. For instance, draw the second circle as a Filled Circle before the drawing the innermost one. Then you draw the inner Circle holding the third pointer button—and thus getting a white one inside the black. Finally draw the outer Circle using the first pointer button (you could actually draw the outer circle at any point). There are many ways to create most pictures.

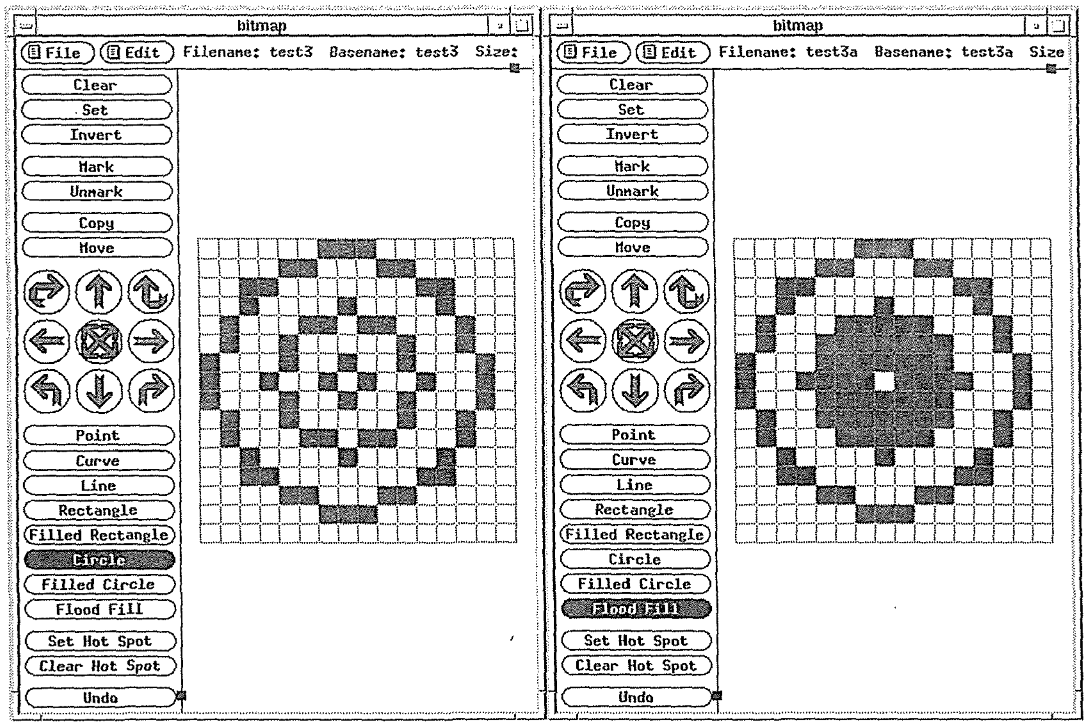


Figure 7-7. Flood Fill

Clear, Set, and Invert

Clear, Set, and Invert work on the entire grid. Note that each command has a keyboard shortcut, or accelerator (the first letter of the command). To use a shortcut, the pointer must rest in the grid or the whitespace surrounding it on the same side of the *bitmap* window.

Clear	Changes all the grid squares to the background color (often white) and sets all bitmap bits to 0. You can perform the same function by typing c while the pointer rests inside the grid or the surrounding whitespace.
Set	Changes all the grid squares to the foreground color (often black) and sets all bitmap bits to 1. You can perform the same function by typing s while the pointer rests inside the grid or the surrounding whitespace.
Invert	Inverts all the grid squares and bitmap bits, as if you had clicked the second pointer button over each square. You can perform the same function by typing i while the pointer rests inside the grid or the surrounding whitespace. In Figure 7-8 we've inverted the concentric circles from the Flood Fill example.

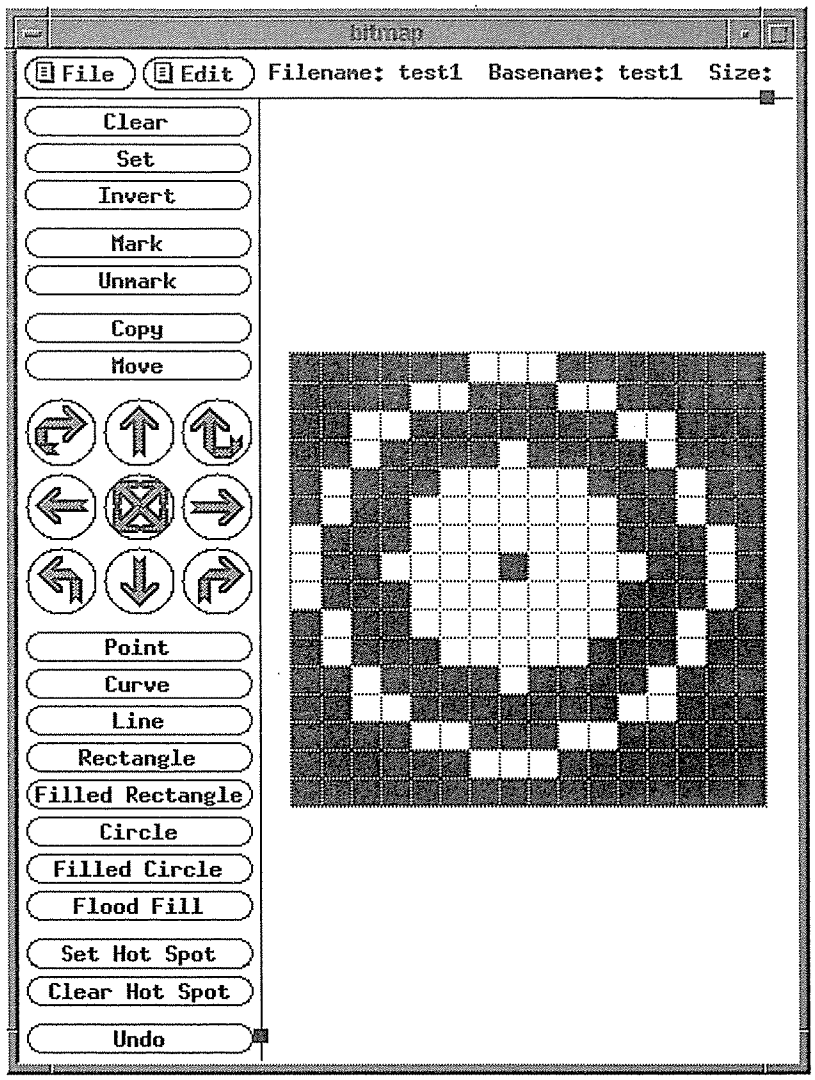


Figure 7-8. Inverting all squares in the grid

Marking an Area for Editing or Pasting

The Mark and Unmark commands are not particularly intuitive, but they are extremely important. When you Mark a part of the grid, many subsequent editing commands apply to that area only. For instance, you might mark a portion of a bitmap and then delete (cut) it. If you mark a part of a bitmap, you can then use the arrow buttons to move that part within the editing area one square at a time.

Note that Mark requires you to select a rectangular area (a line counts); no irregular areas can be marked. The smallest area you can mark is a line composed of two squares.

The most important aspect of marking is that any marked area is available to be pasted from a global buffer. Specifically, the image is copied to the PRIMARY selection, where it is globally available to be pasted via the server. Chapter 5, *The xterm Terminal Emulator*, describes the selection mechanism as it applies to *xterm*, which interprets only "text" selections. However, a selection can be in a variety of formats. The main limitation in the selection mechanism is that copying and pasting must be performed between clients that interpret data in the same format. For our purposes, this means that you can paste part of a bitmap image to a *bitmap* or *xmag* window; you can't paste a bitmap image into an *xterm* window, or paste text into *bitmap*.*

Since there can only be one PRIMARY selection on a display at any time, any area you mark in a *bitmap* window supercedes the previous selection—whether the previous selection is text or graphics. (*Only one area on the display can be marked at any time.*)

To mark part of a bitmap image:

1. Click the first pointer button on the Mark command button. The button displays in reverse video.
2. Move the pointer into the editing area. It changes to a crosshair symbol.
3. Place the crosshair pointer on the upper-left square of the area you want to mark. (Upper left is easy for a righthanded person, but you can actually drag from any corner.) Press and hold down any pointer button. A gray shading covers the square the pointer is on.
4. Drag the pointer away from that square. The area you cover with the pointer will be outlined in gray. In Figure 7-9, we drag the pointer diagonally to the right, highlighting an area 9 squares wide by 11 squares high.
5. When you've marked the area you want, release the pointer button. The entire area will be highlighted in gray.

An area remains marked until you explicitly Unmark it or until you mark another area (or make a selection in any application).

Having a marked area does not interfere with any drawing you do in the window (using the pointer, command buttons such as (Point, Line, etc.). However, if an area is marked, the following commands will act only on that area:

- Copy command button.
- Move command button.
- Cut item on Edit menu (and its keyboard shortcut, Meta-C).
- Copy item on Edit menu (and its keyboard shortcut, Meta-W).
- The arrow command buttons to flip, move, etc., and the corresponding keyboard shortcuts. The center arrow button, which performs the Fold function, is the exception. It acts on the entire bitmap image.

*If you try to paste a text selection into a *bitmap* window, you may cause the client to hang.

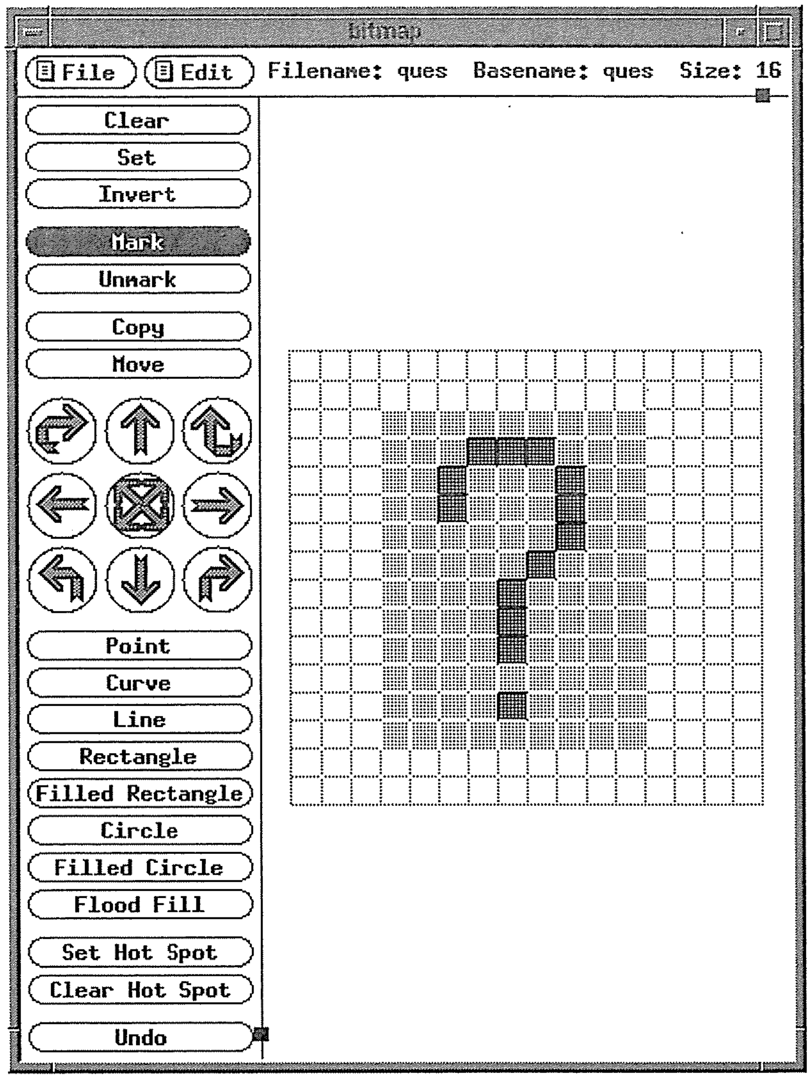


Figure 7-9. Marking an area

Keep in mind that if you select a copy or move and an area is already marked, you will be copying or moving that area. Note, however, that you don't have to select Mark before selecting the Copy or Move command buttons. These functions have "marking" built into them. See "Copying or Moving an Area Within the Bitmap Window" for details. If you want to move part of a bitmap using the arrow command buttons, on the other hand, you *must* Mark it first.

Copying or Moving an Area Within the Bitmap Window

The Copy and Move command buttons allow you to copy or move a rectangular area you select to another position within the current *bitmap* window. (Note, however, that the area you select for the copy or move can be pasted into another *bitmap* window, or into an *xmag* window, using other mechanisms. See "Transferring Bitmap Images Using the Edit Menu" later in this chapter for information about transferring images from window to window.) The Copy and Move command buttons are very useful, if slightly confusing. You perform both functions the same way, but the result is slightly different:

Copy	Makes a copy of a rectangular area and allows you to place it on another part of the grid. (You end up with two copies of that part of the image.)
Move	Moves a rectangular area from one part of the grid to another. (You end up with one copy of that part of the image.)

The first step in either function involves marking the area to be copied or moved. Marking is built into both functions. To Copy or Move part of an image, perform the following steps. (If you've previously marked an area using the Mark command button, you can skip steps 2-5.)

1. Click the first pointer button on the Copy or Move command button, as appropriate. The button displays in reverse video.
2. Move the pointer into the editing area. It changes to a crosshair symbol.
3. Mark the area you want to copy or move. To do so, place the crosshair pointer on the upper-left square of the area in question. (Upper-left is easy for a righthanded person, but you can actually drag from any corner.) Press and hold down any pointer button. A gray shading covers the square the pointer is on.
4. Drag the pointer away from that square. A gray rectangle will surround the area you indicate with the pointer.
5. When you've surrounded the area you want, release the pointer button. The entire area will be highlighted in gray.
6. Then you can move the area or a copy of it, depending on which command you selected initially. To do so, place the pointer on the gray rectangle and hold down any pointer button.
7. Then move the pointer to position the copy/original. An outline of the rectangle follows the pointer's movement. When the outline is positioned where you want it, release the pointer button. The image is redrawn in the place you indicated.

Notice that the area remains highlighted until you select another command. You can continue to move/copy the highlighted area by following steps 6 and 7 above. Note that if you copy an area and place the copy overlapping the original, the next copy will incorporate that image. In other words, when you perform steps 6 and 7, you're copying or moving what you see at the moment (not necessarily the same image you originally marked).

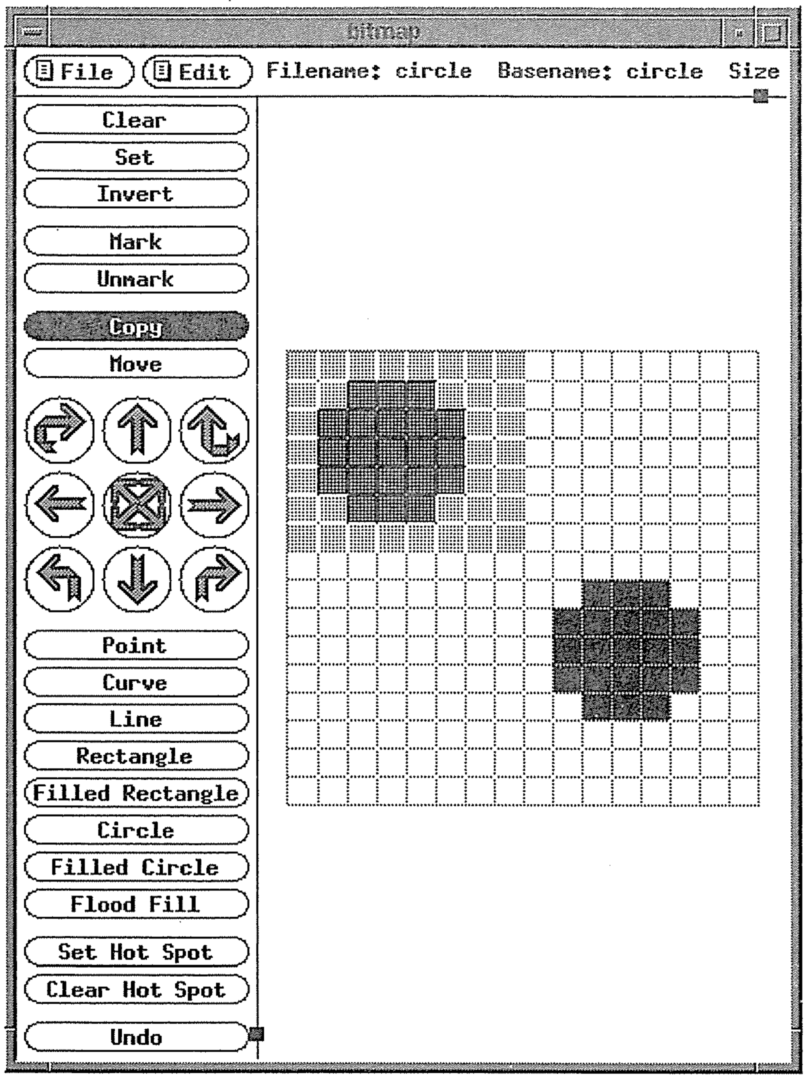


Figure 7-10. Copying an image

Figure 7-11 shows a circle being copied. Remember: if you previously highlighted an area using the Mark command, you can move or copy that area by selecting either Move or Copy (step 1) and then performing steps 6 and 7.

Rotating or Moving with Arrow Buttons

Halfway down in the column of command buttons are 9 circular buttons each marked with some sort of arrow. These buttons correspond to functions that flip or move the bitmap image in various ways. The developer of the *bitmap* client associates each of these buttons with a text name (though obviously the name does not appear on the button). .Figure 7-11 shows the button images on the left and their corresponding "names" on the right.

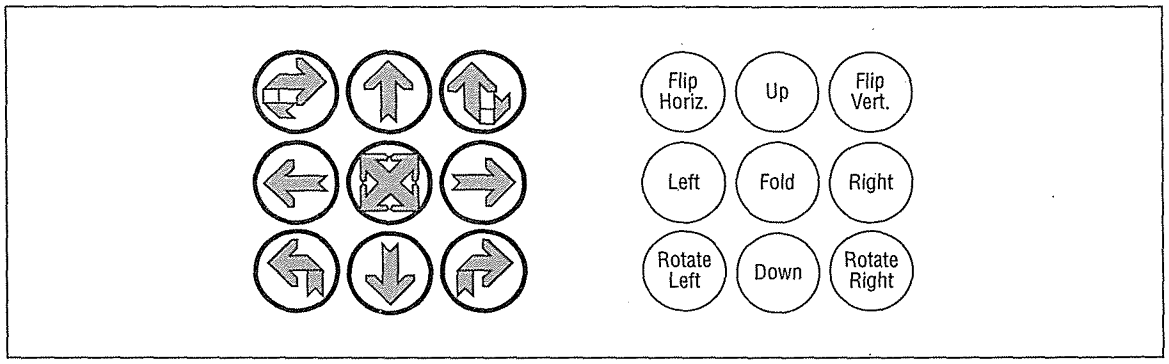


Figure 7-11. Arrow command buttons and their functions

Note that all of the buttons (except Fold) will act on a part of the bitmap, if it has been marked. (See "Marking an Area for Editing or Pasting" earlier in this chapter.)

When you mark an area, the commands to move (Left, Right, Up, Down) do so *only within that area*. Thus, if you have a square 3 × 3 pixels and you want to move it 5 pixels to the right, you must mark an area larger than the square itself.

Note also that to use a keyboard shortcut, the pointer must rest inside the editing grid or the surrounding whitespace (not in the menu bar or the command button column).

Flip Horizontally	Flips the image with respect to the horizontal axis. Pressing h inside the editing area has the same effect.
Up	Moves the image up one pixel (grid square) at a time. Pressing the up arrow on your keypad has the same effect.
Flip Vertically	Flips the image with respect to the vertical axis. Pressing v inside the editing area has the same effect.
Left	Moves the image to the left one pixel (grid square) at a time. Pressing the left arrow on your keypad has the same effect.
Fold	A confusing command. Technically, it "folds" the image so that opposite corners become adjacent, but this explanation is just about as topsy-turvey as the reality. To get a better idea, imagine that the bitmap image is divided into four parts (along the vertical and horizontal axes of the editing area); then the diagonally opposite quadrants are swapped. Since it's much easier to see this for yourself, we provide an

example in Figure 7-12. Note that pressing f inside the editing area has the same effect.

Right	Moves the image to the right one pixel (grid square) at a time. Pressing the right arrow on your keypad has the same effect.
Rotate Left	Rotates the image 90 degrees to the left (i.e., counterclockwise). Thus, selecting this command four times in a row brings the image around 360 degrees, to its original position. Pressing l inside the editing area has the same effect.
Down	Moves the image down one pixel (grid square) at a time. Pressing the down arrow on your keypad has the same effect.
Rotate Right	Rotates the image 90 degrees to the right (i.e., clockwise). Thus, selecting this command four times in a row brings the image around 360 degrees, to its original position. Pressing r inside the editing area has the same effect.

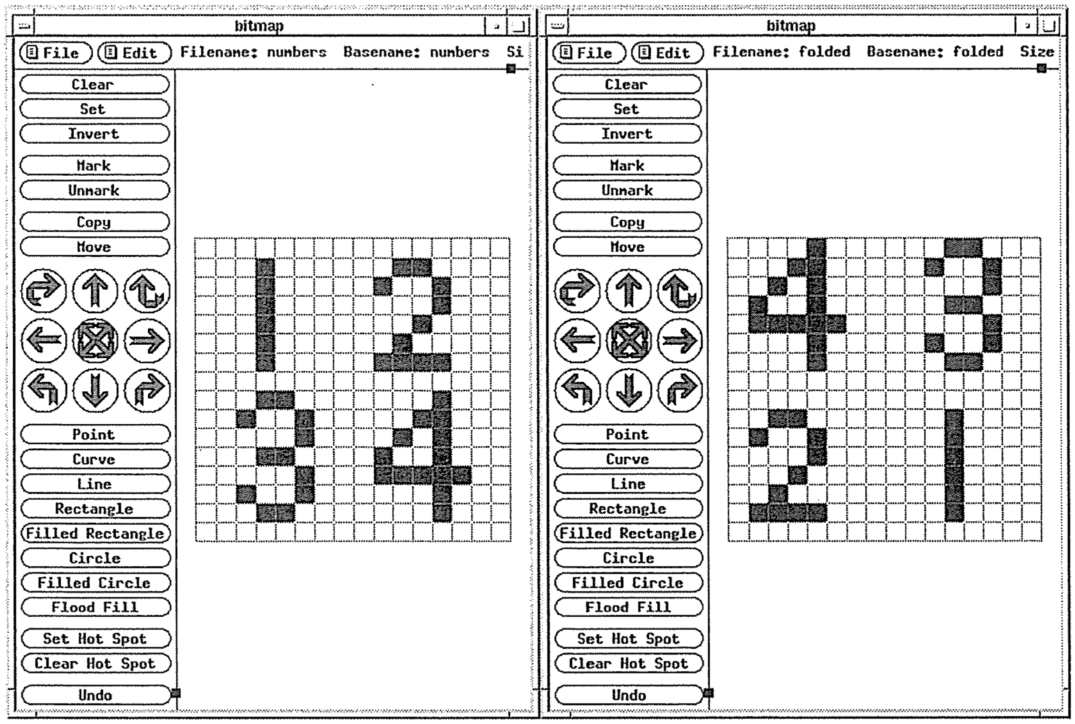


Figure 7-12. Folding an image: before and after

Setting and Clearing a Hot Spot

A *hot spot* is significant if you create a bitmap you want to use as a pointer symbol. If a program is using your bitmap as a pointer, the hot spot indicates which point on the bitmap will track the actual location of the pointer. For instance, if your pointer is an arrow, the hot spot should be the tip of the arrow; if your pointer is a cross, the hot spot should be the intersection point of the perpendicular lines. There can only be one hot spot on a bitmap.

Set Hot Spot	Designates a point on the bitmap as the hot spot.
Clear Hot Spot	Removes a hot spot defined on this bitmap.

To set a hot spot:

1. Click the first pointer button on Set Hot Spot. The button displays in reverse video.
2. Move the pointer to the location of the hot spot. Click the first or second pointer button. When a hot spot is active a diamond (◊) appears in the square.

In Figure 7-13, we set the hot spot of our arrow bitmap to be the tip. Note that the Set Hot Spot button remains active (in reverse video) until you select a mutually exclusive option (any of the drawing commands, Mark, Copy, or Move).

To clear a hot spot:

- Click pointer button one on Clear Hot Spot. The button flashes and the diamond symbol is removed from the bitmap image. Or:
- Click the second or third pointer button on the hot spot.

While Set Hot Spot is selected, the second pointer button toggles the hot spot on and off.

Note that if you use the Move command button to move a bitmap image with a hot spot, the hot spot remains stationary (i.e., it does not move with the image). If you use the arrow buttons to move an image with a hot spot, however, the hot spot will move with the image.

Using the Menus

As mentioned previously, the Release 5 *bitmap* provides two menus that can assist you in creating bitmap images. File menu items help you manage bitmap files, while Edit menu items assist with editing tasks such as cutting and pasting. Both are accessed from the menu bar that spans the top of the application window. To select an item from either menu:

1. Place the pointer on the appropriate menu command button (labeled either File or Edit).
2. Press and hold down the first pointer button. The menu is displayed.
3. Drag the pointer down the menu. A highlight bar tracks the pointer's movement. When you highlight the item you want, select it by releasing the pointer button; then menu is removed.

Each of the menu items has a keyboard shortcut (or accelerator), which appears on the menu following the text label for the item. If you familiarize yourself with these shortcuts, you can invoke any function from the menus without actually having to display them.

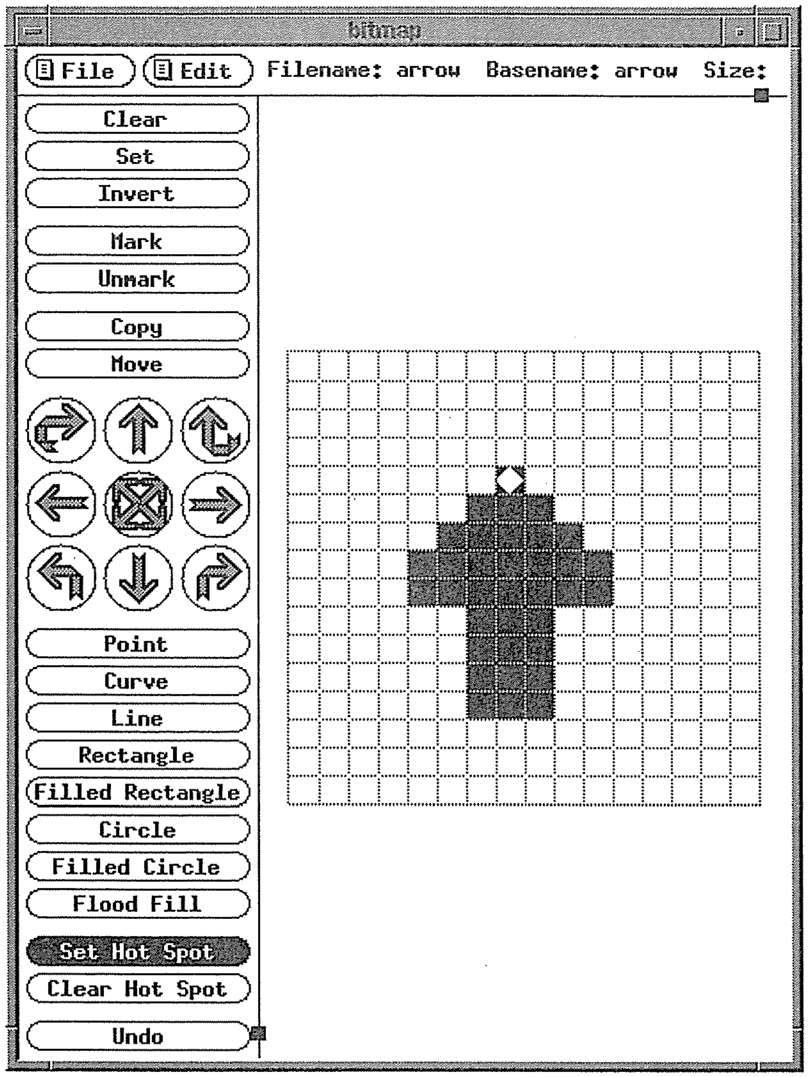


Figure 7-13. Setting a hot spot

Dialog Boxes and Command Buttons

Certain menu selections require you to supply additional information, such as the name of a file, new dimensions for the editing area, etc. In such cases, *bitmap* provides a *dialog box* in which you can enter the information. A dialog box is a small window some applications provide to request information from a user or to notify the user of something. (Figure 7-14 shows the dialog box *bitmap* displays when you try to quit the application without first saving changes.) The dialog box provided by *bitmap* is an X Toolkit feature. It differs somewhat from a dialog box created with the Motif Toolkit. See "Dialog Boxes and Push Buttons" in Chapter 9, *Working with Motif Applications*, for information about Motif dialogs.

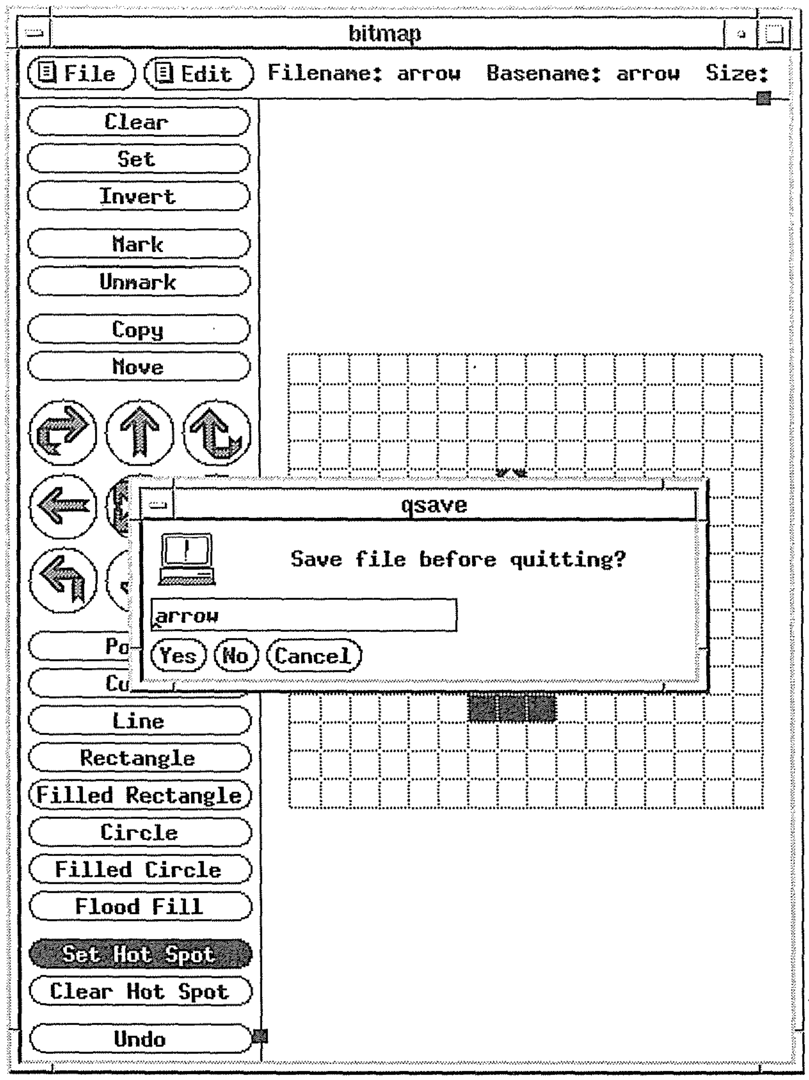


Figure 7-14. Bitmap window with quit dialog box

A dialog box created using the X Toolkit typically has three elements: it always has the first element, and may or may not have the second and/or third elements in this list:

- A prompt that identifies the purpose of the widget. For example, if you try to quit *bitmap* without having saved changes to the current file, the dialog prompts: "Save file before quitting?" If you want to Load another file, *bitmap* prompts: "Load file:" and you respond with the filename. Some prompts merely provide information to the user.

- An area in which you can type your response (if required).
- Command buttons that allow you either to acknowledge the prompt's message, or to confirm or cancel the dialog input, as appropriate.

When a dialog box pops up, the pointer is automatically redisplayed in the box (this is known as "warping the pointer"), and the box is given the input focus, so you can type. A dialog box is usually a pop-up window that disappears after the required information is provided.

Each command button in the box is itself a widget. A command button is usually a rectangle or oval that contains a text label. When you click a on a command button, some action (presumably indicated by the label) is performed by the program. The X Toolkit command button is analogous to the push button provided by the Motif Toolkit, which is described in Chapter 9.

To respond to the *bitmap* quit dialog box, click the first pointer button on the appropriate command button. You can select the first command (Yes) simply by pressing Return.

Notice that the dialog box contains a small text window displaying the name of the file. If you like, you can edit this filename before saving. The text window is an instance of the Athena Text widget, the editing commands for which are described in "The xedit Text Editor" in Chapter 8, *Other Clients*. The caret symbol (^) at the beginning of the filename is the text cursor.

The File Menu

The File menu helps you manage bitmap files. Using File menu items you can:

- Dynamically load another bitmap file into the editing window (Load).
- Insert another bitmap image into the current bitmap file (Insert).
- Save the current file (Save or Save As) and start editing a new one (New).
- Change the current filename (Filename).
- Change the dimensions of the grid (Resize).

Table 7-1 lists the File menu items, their keyboard shortcuts, and gives a brief description of what each item does. You can probably figure out how to use most of the items simply by trying them. Following the table is a tutorial that should help you use the somewhat less intuitive Insert command, which lets you insert another bitmap image into the current file.

Table 7-1. File menu items

Menu Item	Keyboard Shortcut(s)	What Item Does
New	Control-N	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	Control-F	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.
Insert	Control-I	Inserts a bitmap file into the image currently being edited. See the next section, "Inserting a File," for instructions.
Save	Control-S	Saves the current image using the filename in the menu bar.
Save As	Control-W	Saves the current image but prompts for the filename. That name is subsequently displayed in the menu bar.
Resize	Control-R	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	Control-X	Changes the dimensions of the grid to match dimensions you supply (`widthxheight`) 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, the grid and the image will both be doubled. Specifying a smaller grid may cause part of the current image to be truncated.
Filename	Control-E	Lets you change the filename of the current file without changing the basename (which appears in the header lines of the file) or saving the file.
Basename	Control-B	Lets you change the basename (if you want one different from the filename). The basename appears in the bitmap file, as part of the C code.
Quit	Control-C, q, Q	Exits the application. If changes have been made and not saved, a dialog box will ask whether to save before quitting.

Inserting a File

To insert a bitmap file into the image you're editing, select Insert from the File menu. A dialog box will prompt you for a filename, as in Figure 7-15.

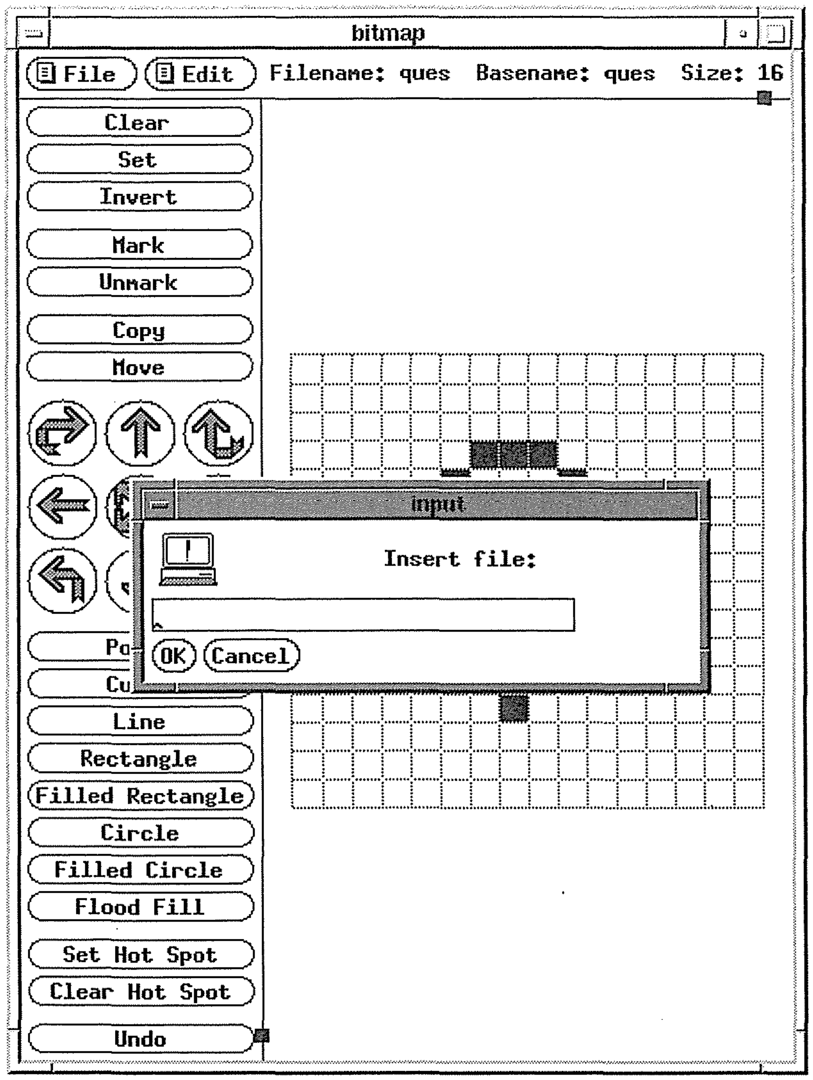


Figure 7-15. Dialog box requests name of file to insert

The pointer will be warped to the dialog box, enabling you to enter the path of the file you want. Use the editing commands for the Athena Text widget (described in the section "The xedit Text Editor" in Chapter 8, *Other Clients*). Once you type the filename, click on OK or press Return to request the file.

If the file exists, the pointer will change to the crosshair symbol. Move the pointer onto the grid and press and hold down the first pointer button. You should see a gray outline that represents the image to be inserted. Drag the image to the position you want and release the pointer button. The image is inserted into the current file. In Figure 7-16, we've inserted a large circle into the arrow file.

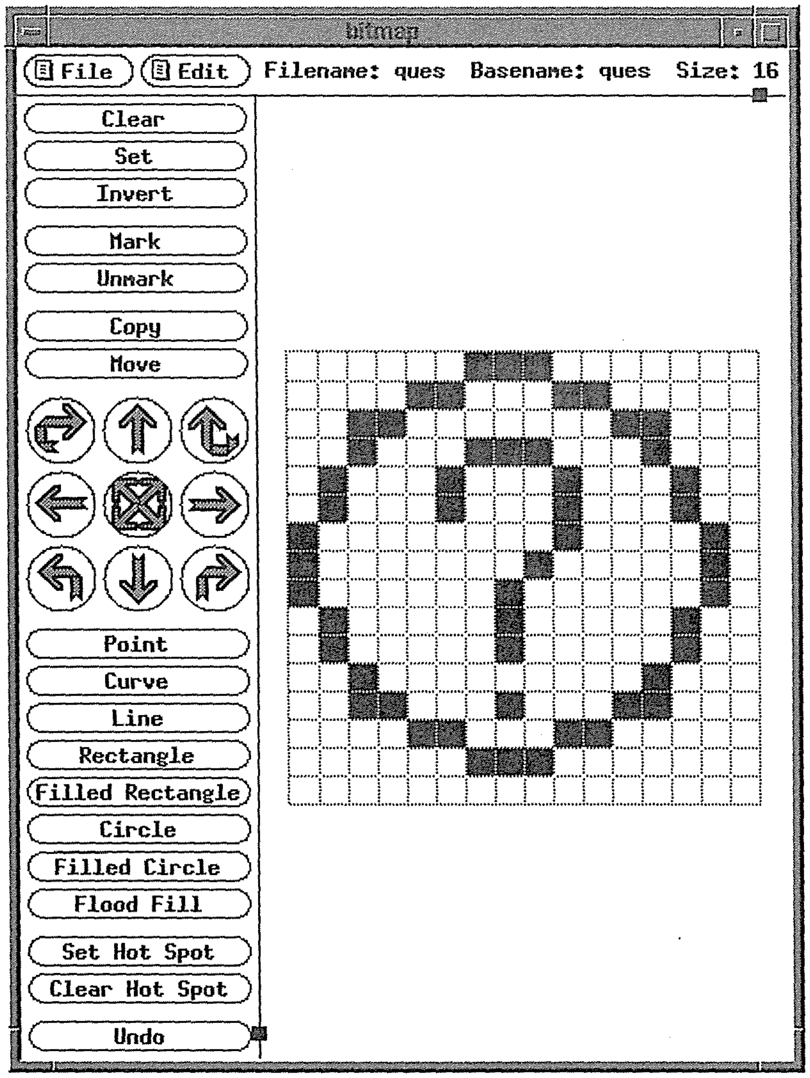


Figure 7-16. Image is inserted into open file

The Edit Menu

The Edit menu is divided into two sections by a horizontal line. The top portion contains items that determine characteristics of the editing area, such as its dimensions, whether or not grid lines are used, etc. Most of these items turn a characteristic on and off (i.e., the items are toggles). The bottom portion contains commands to Copy, Cut, and Paste images within the same *bitmap* window, between *bitmap* windows, or between *bitmap* and *xmag*.

Table 7-2 lists the Edit menu items, their keyboard shortcuts, gives a brief description of what each item does. You can probably figure out how to use most of the items simply by trying them. Following the table is a discussion of "Transferring Bitmap Images Using the Edit Menu."

Table 7-2. Edit Menu Items

Menu Item	Keyboard Shortcut(s)	What Item Does
Image	Meta-I	Displays a window showing what the bitmap being edited looks like at its actual size (both as it appears and in reverse video). Clicking the first pointer button on this window pops it down.
Grid	Meta-G	Toggles grid lines. 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 it on by selecting this menu item.
Dashed	Meta-D	Toggles stippling in the grid lines. On by default when the grid lines are activated.
Axes	Meta-A	Toggles diagonal axes. The axes simply assist in drawing; they are not part of the image. Off by default.
Stippled	Meta-S	Toggles a stipple pattern to be used for highlighting within the editing area. (This stipple is a subtle shading; if toggled off, marking is done in the foreground color.) On by default.
Proportional	Meta-P	Toggles proportional mode which forces proportional grid squares, regardless of the dimensions of the *bitmap* window. On by default.
Zoom	Meta-Z	Toggles zoom mode, which focuses in on a marked area of the image. (You can mark before or after selecting Zoom.)
Cut	Meta-C	Cuts the contents of any marked area into the application's local buffer. The marked area is deleted from the current image, but is available to be pasted from the local buffer. (If this was the last area marked, it is also available to be pasted into other applications via a global buffer.)
Copy	Meta-W	Copies the contents of any marked area into the applications's local buffer. The marked area remains a part of the current image and is also available to be pasted from the local buffer. (If this was the last area marked, it is also available to be pasted into other applications via a global buffer.)

Table 7-2. Edit Menu Items (continued)

Menu Item	Keyboard Shortcut(s)	What Item Does
Paste	Meta-Y, Control key and mouse button click	Pastes the contents of the global buffer (the marked area in any *bitmap* or *xmag* application), or if the global buffer is empty, the contents of the local buffer, into the current image. 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.

Transferring Bitmap Images Using the Edit Menu

Though it is hardly obvious, the Copy and Cut items on the Edit menu provide more powerful image transferring capabilities than the Copy and Move command buttons. As we've seen, the Copy and Move command buttons allow you to transfer images within the current *bitmap* window. These buttons copy the part of the image you select into a *local* buffer (available only within the particular window).

However, in addition to this local buffer, *bitmap* works with a *global* buffer. Specifically, a "marked" part of an image becomes the PRIMARY selection—and is available to be pasted to any *bitmap* or *xmag* window. Moreover, when you mark an area and then Copy or Cut it using the Edit menu (or the keyboard shortcuts for those items), the image is also available to be pasted globally.* Admittedly, this business of transferring images can seem confusing, but it's fairly easy in practice. Let's consider an example and then try to clarify the precedence rules for pasting from the global and local buffers.

First, let's run two *bitmap* windows, one with a question mark image and the other an exclamation point, as in Figure 7-17.

*You *can* use these items to copy or cut images within the current window—but this is not their primary intent.

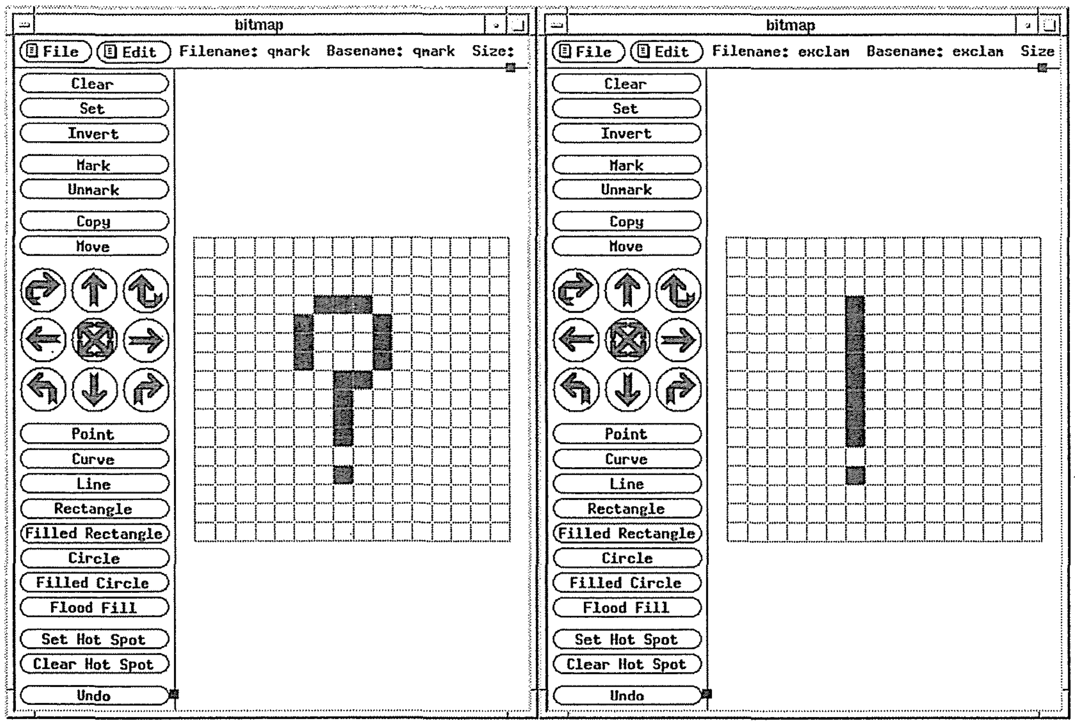


Figure 7-17. Two bitmap windows

Say we want to copy the exclamation mark from the right *bitmap* window into the left window, so that it appears after the question mark. There are a few ways to do this. We could:

1. Mark the exclamation using the procedure outlined for the Mark command button. While the exclamation is highlighted in this way, it is available to be pasted via the global buffer. In Figure 7-18, we've marked an area encompassing the exclamation and one additional pixel on all sides. (The entire area will be pasted.)

2. To paste the marked area into the question mark window, select Paste from the Edit menu in that window. Then move the pointer into the editing area. The crosshair pointer represents the upper-left pixel of the image to be pasted. Place the image where you want and click the first pointer button. The image is pasted, as in Figure 7-18.

Rather than selecting the Paste menu item, you could instead direct the focus to the question mark window and use either of the two keyboard/pointer shortcuts for the item: Meta-Y or Control and any mouse button click. Note that in the latter case, the click places the image. Until you mark another image, invoking the paste function in any *bitmap* or *xmag* window will paste the exclamation.

There are other ways to place the exclamation image in the question mark window. You could also highlight the exclamation image, then select either Copy or Cut from the Edit menu in the same window. (These items are only available for selection when a image is marked in the window.) Obviously, the former menu item makes a copy of the marked image, while the latter makes a copy but deletes the original from the window. In both cases, the image is available globally. Once you've selected Copy or Cut from the Edit menu, you can transfer the exclamation by invoking the paste function in any *bitmap* or *xmag* window.

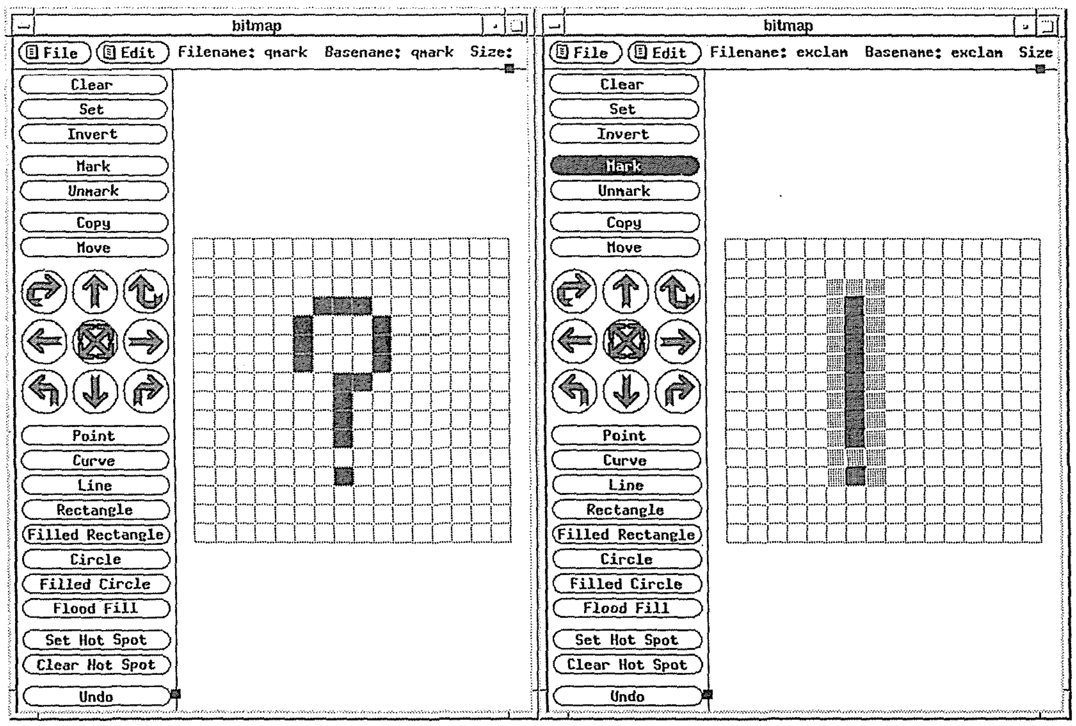
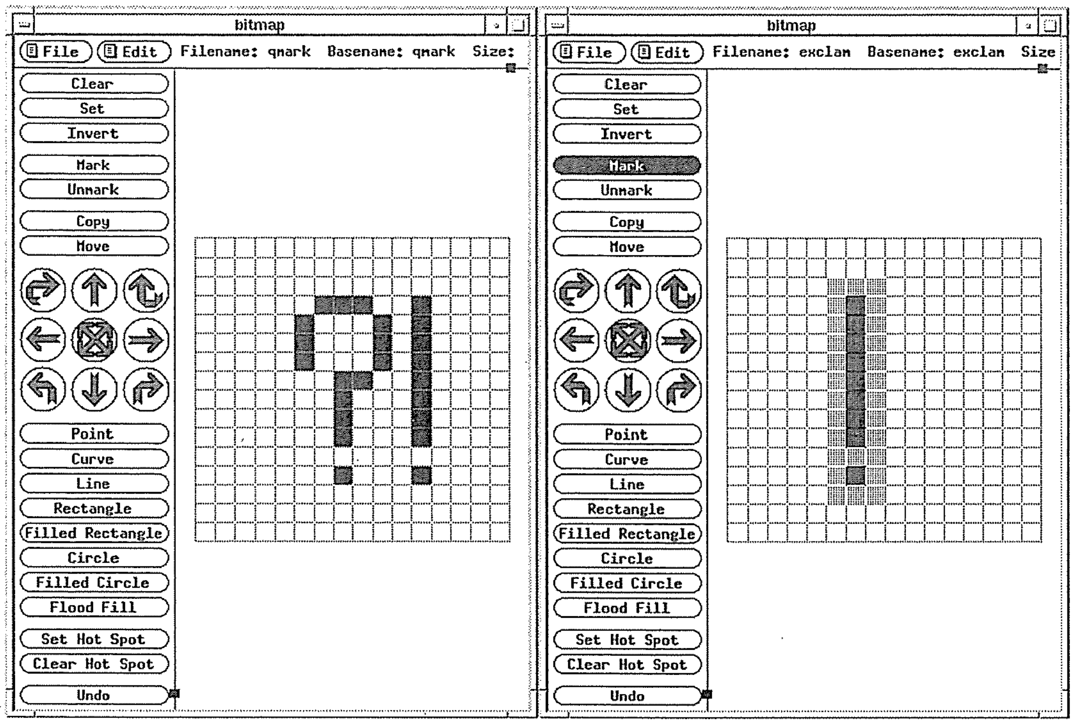


Figure 7-18. Marking the exclamation so that it can be pasted

If you run multiple *bitmap* windows and cut and paste frequently, the contents of the global and local buffers will often be different. In trying to determine what image you'll be pasting, the following rules of precedence apply:

1. If an image is currently marked (highlighted) in any *bitmap* window, that image will be copied to any *bitmap* or *xmag* window in which you invoke a paste command.
2. If no image is currently marked, the image that is pasted will be: the last image cut, copied, or pasted in any currently running *bitmap* window (using the analogous Edit menu options or their keyboard accelerators); or the last image selected in a currently running *xmag* window.
3. Whatever image you Copy or Move using the command buttons can only be pasted within the same *bitmap* window, *unless it remains highlighted*. In this case, the image is available via the global buffer and will be pasted according to rule 1 above.


Filename: qmark Basename: qmark Size:

Filename: exclam Basename: exclam Size

Figure 7-19. Pasting the exclamation next to the question mark

As we'll see in the following sections, the *xmag* client can also access the global image buffer.

Magnifying Portions of the Screen: xmag

The *xmag* client enables you to magnify a portion of the screen. The close-up look *xmag* affords can assist you in creating and editing bitmaps and other graphic images. You can also copy and paste images between *xmag* windows or between *xmag* and the *bitmap* editor.

One instance of the *xmag* program lets you magnify several different areas of the screen and display the images either sequentially (in the same window) or concurrently (in multiple windows). In addition to giving you a look at a magnified bitmap image, *xmag* also provides certain information about the individual pixels in the image.

The Release 5 version *xmag* is clearly intended to be used in concert with the *bitmap* editor. (Previous incarnations of the program were not integrated with *bitmap*.) Of course, application developers may also find it useful when working with more sophisticated graphics programs.

When might you use *xmag*? Say you're running a program that creates a special image on the root window and you'd like to create a *bitmap* file of a part of that image. You can display a magnification of the image you want with *xmag*. Then you can either try to recreate the image by editing in an open *bitmap* window or copy the *xmag* image and paste it into the *bitmap* editor.

xmag is also useful for capturing a screen image that can then be saved as a window dump file with the *xwd* client (described in Chapter 8, *Other Clients*). *xwd* makes a dump file of a single window at a time (though that window may be the root). In some cases, however, you might want to make a dump file of a portion of a window or of multiple windows. *xmag* allows you to magnify any portion of the screen; thus the *xmag* window can capture portions of several windows at once. You can then use *xwd* to make a dump file of the *xmag* window. (Since *xmag* is intended to magnify, if you want the window image to be the actual size, you must specify that no magnification is performed. To do this, run *xmag* with the option `-mag 1`. See the *xmag* reference page in Part Three of this guide for more information.)

Selecting an Area to Magnify

If you invoke *xmag* without options, you can interactively choose the area to be magnified (the *source* area). At the command line, type:

```
% xmag &
```

The pointer changes to an upper-left corner symbol from which extends a small, hollow square with a wavering border. (By default, the square is 64 pixels on each side.) Move the corner cursor, placing the square over the area you want to magnify, and click the first pointer button.

The image inside the hollow square is magnified and displayed in the *xmag* window. Unless you specify a location using `-geometry`, *mwm* places the *xmag* window in the upper-left quadrant of the display. An *xmag* window, containing a magnified bitmap image, is shown in Figure 7-20.

The default size *xmag* window shows an area 64 pixels square, magnified five times; thus, the image itself is 320 pixels on each side. This magnification enables you to see the individual pixels, which are represented by squares of the same color as the corresponding pixels in the source image.

Rather than use the default source area and magnification, you can specify other values on the command line. See the *xmag* reference page in Part Three of this guide for a complete list of options.

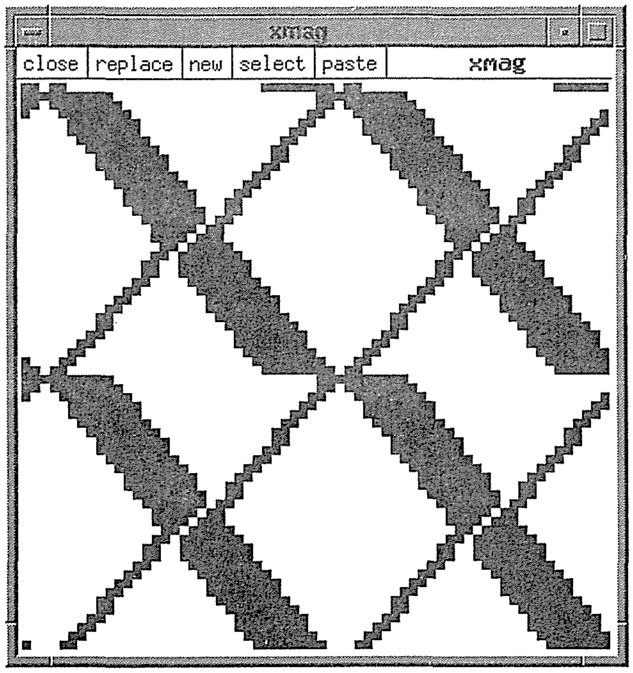


Figure 7-20. xmag window displaying magnified screen area

xmag Command Buttons

The Release 5 version of *xmag* offers many improvements over previous versions, the most obvious being the five command buttons that appear above the magnified image (to the right of the string "xmag"). Once you understand what the command buttons do, you should have you a fairly good idea of *xmag*'s capabilities. You invoke a command button by placing the pointer on it and clicking the first pointer button. When you place the pointer on a command button, the button will be highlighted by a black border; when you select it, the button will flash in reverse video.

Magnifying A Different Source Area: replace or new

If you want to magnify another portion of the screen using the same source area size and magnification factor, you do not have to start *xmag* again. You can choose another source area and either display it in the same *xmag* window or open an additional window in which to display it, using the replace and new command buttons, respectively.

To replace the image in the current window with another magnified image:

1. Place the pointer on the replace command button and click the first pointer button (the replace command button will flash in reverse video). Once again, the pointer becomes an upper-left corner bordering a hollow square.

2. Choose a source area in the manner described in the section "Selecting an Area to Magnify."
3. The *xmag* window is redrawn to display the new magnified image.

As an alternative, you might want to keep the first *xmag* image, but also magnify another area of the screen. In order to display a second *xmag* window:

1. Place the pointer on the new command button and click the first pointer button. The pointer becomes an upper-left corner bordering a hollow square.
2. Choose a source area in the manner described earlier.
3. The source area is magnified and displayed in a second *xmag* window.

You can select any number of source areas during a single *xmag* session.

Copying and Pasting Images: select and paste

You can copy the image displayed in an *xmag* window and paste it in another *xmag* window—or in a *bitmap* window. (As we've seen, you can paste images between *bitmap* windows; but you can also paste an image from *bitmap* into *xmag*.)

To copy the contents of *xmag*, click on the select command button. The *xmag* select function is analogous to "marking" an image in a *bitmap* window. The image is copied to the PRIMARY selection where it is globally available via the server.

There will be no indication that the image has been copied into memory, although the select button will flash in reverse video when you click on it. However, if you then move the pointer to *another xmag* window and click on paste, the image from the first window will be displayed in it.

Think of the select and paste buttons as accessing the same invisible graphics clipboard—which has room for only one image. The *bitmap* copy and paste mechanism accesses the same space in memory. Thus, you can select images from any *xmag* or *bitmap* window (by the mechanism appropriate to the particular client), but only the last image you copy remains in memory to be pasted elsewhere.

A note about pasting an *xmag* image into a *bitmap* window: the default *xmag* image represents a 64 × 64 pixel square while the default *bitmap* image is 16 × 16 pixels. If you intend to transfer images between these clients, it's a good idea to specify comparable dimensions. If you have the screen space, you might run:

```
% bitmap -size 64x64 &
```

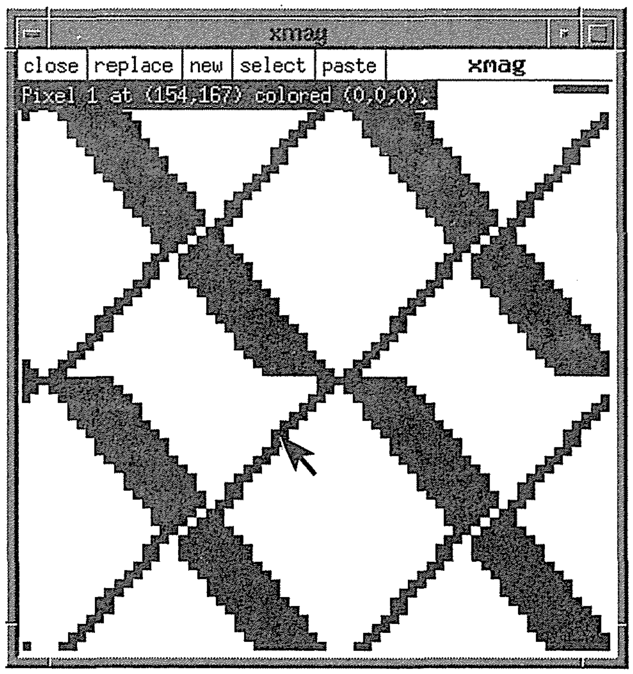
However, this is a very large window. You can make the *bitmap* window smaller by specifying dimensions for the individual editing squares (using the `-sw` and `-sh` options), or you can specify another size for the *xmag* source area:

```
% xmag -source 16x16 &
```

Regardless of the area you want to work with, the important thing to remember is to keep the dimensions comparable.

What xmag Shows You

xmag enables you to determine the x and y coordinates, bitmap bit setting, and RGB color value of every pixel in the *xmag* window. (See Chapter 12, *Specifying Color*, for a discussion of the RGB color model.) If you move the pointer into the *xmag* window, the cursor becomes an arrow. Point the arrow at one of the magnified pixels and press and hold down the first pointer button. A banner across the top edge of the window displays information about the pixel, as shown in Figure 7-21.



Pixel 1 at (154,167) colored (0,0,0).

Figure 7-21. Displaying pixel statistics with pointer in xmag window

The banner displays the following information about the specified pixel:

- The positive x and y coordinates relative to the root window. These are the coordinates of the original image—not of the *xmag* window.
- The bitmap bit setting. This is either 0 if the pixel is in the background color or 1 if the pixel is in the foreground color.
- The RGB value. This is a 16-bit value. The RGB specification is in three parts (of four hexadecimal digits each), corresponding to the three primaries in the RGB color model.

If you are trying to create a graphic image on a grid (such as the *bitmap* client provides), the x and y coordinates of each pixel can be especially useful. Also, the 16-bit RGB value specifies the color of each pixel with incredible precision. Depending on the number of colors available on your display, you can learn to use RGB values to specify an enormous range of colors. (Keep in mind, however, that RGB colors are not portable. See Chapter 12, *Specifying Color*, for more information.)

xmag provides these pixel statistics dynamically. If you continue to hold down the first pointer button and drag the pointer across the window, the banner will display values for each pixel as the pointer indicates it.

Note that if you select a pixel near the top edge of the window, the banner will appear across the bottom edge. Otherwise, the banner would obscure the pixel you are pointing at.

Removing an xmag Window: close

To remove a magnification window, either:

- Place the pointer on the close command button and click the first pointer button; or
- Type q, Q, or Control-C while the pointer rests in the window. Note that the window must have the input focus.

These are "kind" methods for removing a window, which allow all relevant processes to finish.

If *mwm* is running, you could also remove a single window by double-clicking on the Window Menu button on the *mwm* frame; or by selecting the Close item from the Window Menu. You can remove *all xmag* windows (started under the same instance of the program) using the *xkill* client. These three methods all involve "killing" a program, which can adversely affect underlying processes. Refer to the section on *xkill* in Chapter 8, *Other Clients*, for a more complete discussion of the hazards of killing a client.

Creating a Bitmap from a Cursor

The *atobm* and *bmtoa* clients allow you to convert *a*rrays (of ASCII characters) *to b*it*m*ap files and to convert *b*it*m*ap files *to a*rrays. These clients are commonly used to facilitate printing: a bitmap file that is converted to ASCII text can be printed more readily and can also be included in standard ASCII text files. Once converted to ASCII, bitmap files can also be more quickly copied or mailed to other directories or systems, where they can be used in ASCII format or converted back to bitmap format.

Among their uses, the *bmtoa* and *atobm* utilities make it possible to convert a character from a font, such as the cursor font, to the *bitmap* file format. Once converted, the file can be edited using the *bitmap* client and used as you would any other bitmap file: specified as the root window pattern (with *xsetroot*), etc.

When a bitmap file is converted to ASCII text, it is in the form of an array consisting of two types of characters. (An array is a number of elements arranged in rows and columns; it is sometimes called a matrix.) One character represents set or filled squares of the bitmap (bitmap bit 1) and the other character represents empty squares (bitmap bit 0). By default, the number sign character (#) represents filled squares and the hyphen (–) represents empty squares. Figure 7-22 shows the British pound sign character of the 9 × 15 font (in the *misc* directory) as an array of these ASCII symbols.

```
---###-
--#---#
--#----
#####--
--#----
-####--
#-#--##
-#-----
```

Figure 7-22. ASCII array representing the British pound sign

As you can see, the array is a rectangle. In a sense, the array is similar to the *bitmap* grid. (You can edit or create the array using an ASCII text editor, as long as you use the standard two characters and keep the array rectangular.)

To convert the Gumby character of the cursor font to a bitmap, the first thing you must do is display the cursor font as ASCII text. This can be done with the *showfont* client, which allows you to display the contents of a font file (with a *.pcf* extension), if the file is accessed via a font server. (See "Using the Font Server" in Chapter 6, *Font Specification*, for a discussion of the font server and the *showfont* client.)

To display the cursor font with each character represented as an array, use *showfont* (giving the required `-server` option to identify the server and the font name as an argument) and redirect output to a file called */tmp/cursor.array*:

```
% showfont -server server_name cursor > /tmp/cursor.array
```

In this case, we've used the alias for the cursor font, rather than the actual (lengthy) font name.

The *cursor.array* file contains information about the font and an array for each character. Using your ASCII text editing program, edit the file, writing the Gumby array to another file called */tmp/gumby.array*. (Appendix D, *Standard Cursors*, saves us some scanning by telling us that Gumby is character number 56, so we search *cursor.array* for 56.) The Gumby array is pictured in Figure 7-23.

```
--######--------
---#----#-------
##--#----#------
###-#-#-#-#-----
##--#-----#-----
##--#-###-#-----
#####-----####--
--###-----######
----#-----#--###
----#-----#--###
----#--#--#-####
----#--#--#--###
----#--#--#-----
---#---#---#----
--#----#----#---
--#####-#####---
```

Figure 7-23. /tmp/gumby.array

You can then use the *atobm* client to convert this array to a bitmap. Use the *gumby.array* file as an argument and redirect the output to a bitmap file:

```
% atobm /tmp/gumby.array > /tmp/gumby.bitmap
```

Figure 7-24 shows the Gumby bitmap. As you can see from the bitmap, the Gumby character of the cursor font is considerably smaller than the Gumby we created (Figure 7-14) with *bitmap*.

If you want, you can then edit the *gumby.bitmap* file using the *bitmap* client.

If you specify the unedited bitmap as the root window pattern, you'll notice that there is virtually no space between the Gumby figures. This is because the array file had no extra hyphens (representing empty *bitmap* squares) padding it. If you want, you can add some hyphens to the *gumby.array* file (keeping the image symmetrical) and then use *atobm* to create a more padded version of the bitmap. Figure 7-25 shows the *gumby.array* file after it was padded with hyphens.

See the *bitmap* reference page in Part Three of this guide for more information on the *atobm* and *bmtoa* conversion clients.

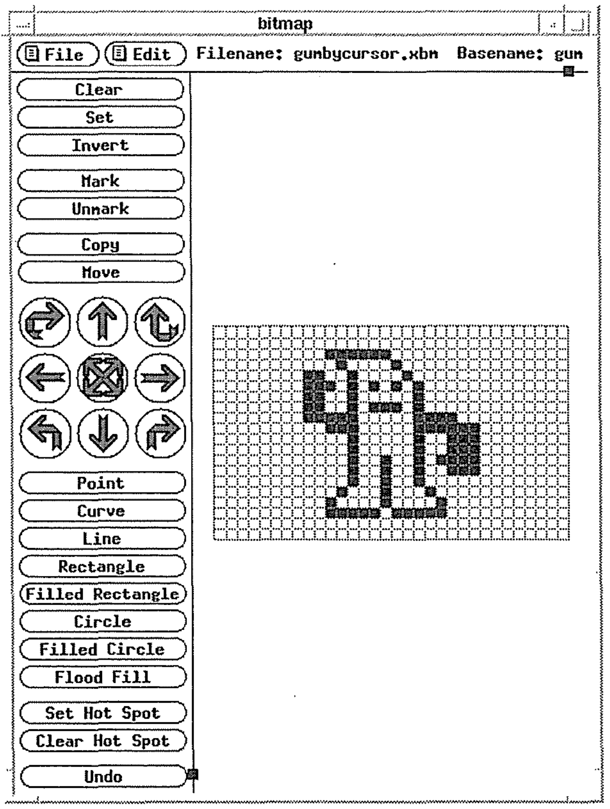


Figure 7-24. Bitmap of the Gumby cursor

```
--------------------------------
--------------------------------
----------######----------------
-----------#----#---------------
--------##--#----#--------------
--------###-#-#-#-#-------------
--------##--#-----#-------------
--------##--#-###-#-------------
--------#####-----####----------
----------###-----######--------
------------#-----#--###--------
------------#-----#--###--------
------------#--#--#-####--------
------------#--#--#--###--------
------------#--#--#-------------
-----------#---#---#------------
----------#----#----#-----------
----------#####-#####-----------
--------------------------------
--------------------------------
```

Figure 7-25. gumby.array padded by hyphens

The Portable Bitmap Toolkit

The Portable Bitmap Toolkit (in the user-contributed distribution) provides dozens of utilities for converting graphics files to and from portable formats. Developed by Jef Poskanzer, the Toolkit is composed of four parts, three of which correspond to a particular portable format:

- PBM: utilities to convert files to and from portable bitmap format.
- PGM: utilities to convert files to and from portable graymap format (grayscale images).
- PPM: utilities to convert files to and from portable pixmap format (color images).

The fourth part of the toolkit, PNM, provides utilities to manipulate images in any of the three formats. For example, the program *pnmenlarge* enlarges a portable "anymap" by a factor you supply. *pnminvert* inverts an image in any of the three portable formats.

The available utilities and the conversions they perform are summarized in the *README* file in the source directory. Table 7-1 lists some representative conversion utilities and their functions.

Table 7-3. Some PBM Toolkit Conversion Utilities

Utility	Converts
giftoppm	GIF to portable pixmap
ppmtogif	Portable pixmap to GIF
ppmtopgm	Portable pixmap to portable graymap
fstopgm	Usenix FaceSaver file to portable graymap
pnmtops	Portable anymap to Encapsulated PostScript
pgmtopbm	Portable graymap to portable bitmap
pbmtomacp	Portable bitmap to MacPaint
macptopbm	MacPaint to portable bitmap
pbmtoxbm	Portable bitmap to X11 bitmap
pbmtox10bm	Portable bitmap to X10 bitmap
xbmtopbm	X10 or X11 bitmap to portable bitmap
pnmtoxwd	Portable anymap to X11 window dump
xwdtopnm	X10 or X11 window dump to portable anymap

As the table indicates, some of the available utilities come in pairs—they can be used to convert a file to a portable format and back to its original format again. (The table also includes a group of three related utilities to convert X10 and X11 bitmaps to portable bitmaps and back again.)

Certain conversions can only be performed in one direction. For example, you can convert a portable graymap to a portable bitmap (using *pgmtopbm*), but you can't convert a bitmap to a graymap. The one-way conversions generally involve changing a file to a simpler format.

You'll probably be most interested in converting graphics files to formats suitable for use with X, namely X bitmaps or window dump files. Keep in mind that a portable bitmap has a different format than an X bitmap. The program *pbmtoxbm* converts a portable bitmap to a bitmap compatible with X11.

The conversions you may want to perform can be simple (directly from one format to another) or complex (through several intermediate formats). An example of a simple conversion is changing a portable pixmap to a portable graymap using *ppmtopgm*:

```
% ppmtopgm pixmap > graymap
```

The PBM Toolkit source directory includes a file called *TIPS* that provides helpful hints on using the utilities. Based on these suggestions, we performed a fairly complex conversion: a Usenix FaceSaver image to a bitmap suitable for use with X. The following command performed the conversion on the file *myface* to create *myface.bitmap*:

```
fstopgm myface | pnmenlarge 3 | pnmscale -yscale 1.125 | pgmnorm |\
        pgmtopbm | pbmtoxbm > myface.bitmap
```

Notice that this particular conversion requires six utilities! This procedure is by no means intuitive. We relied heavily on the *TIPS* provided.

The six conversions performed are:

- Convert FaceSaver image to portable graymap (`fstopgm`).
- Enlarge a portable anymap three times (`pnmenlarge 3`).
- Scale pixels in y dimension; x dimension is adjusted accordingly (`pnmscale -yscale 1.125`).
- Normalize contrast of portable graymap (`pgmnorm`).
- Convert portable graymap to portable bitmap (`pgmtopbm`).
- Convert portable bitmap to X11 bitmap (`pbmtoxbm`).

Be aware that the command:

```
pnmscale -yscale 1.125
```

may not be necessary on all systems or the necessary arguments may vary. If you omit *pnmscale* and the command is necessary, the system should return a message to that effect and also tell you what arguments to use.

The possible uses of the PBM Toolkit programs and the ways in which they can be combined are extremely varied. You'll have to do some experimenting. To orient yourself, read the files *README*, *TIPS*, and *FORMATS* in the source directory. The source directory also includes reference pages for each utility.

8

Other Clients

This chapter gives an overview of other clients available with X, including window and display information clients, printing utilities, the xkill *program, and several "desk accessories." It also describes how to use three public-domain clients that can assist you in creating and specifying colors for your display:* xcol, xcoloredit, *and* xtici.

In This Chapter:

☞

8 Other Clients

In addition to *xterm*, the MIT distribution includes many other clients. This chapter discusses some of the more useful clients, grouped according to basic functionality:

- Program to redraw the screen: *xrefresh*.
- Desk accessories: *xclock*, *oclock*, *xcalc*, *xbiff*, *xload*, and *xman*.
- Text editor: *xedit*.
- Printing utilities: *xwd*, *xpr*, and *xdpr*.
- Program to remove a client window: *xkill*.
- Window and display information programs: *xwininfo*, *xlsclients*, and *xdpyinfo*.
- Alternative window managers and other user-contributed clients, including: a program to display available colors and change color preferences: *xcol*; two programs to create or "mix" your own colors, *xcoloredit* and *xtici*.

Most sections in this chapter are intended to acquaint you with the major features of some of the available clients. Additional detailed information is provided on the reference pages for each client in Part Three of this guide.

Many of the standard clients have been written (or rewritten) with a programming library called the X Toolkit. As explained in Chapter 1, the X Toolkit includes a set of predefined components (or widgets), known as the Athena widget set. Widgets make it easier to create complex applications; they also ensure a consistent user interface between applications.

Although most of the standard clients described in this guide were originally written before the X Toolkit was fully developed, many have since been rewritten to use the Toolkit. In discussing various clients in this chapter, we'll point out some of the features attributable to the X Toolkit. (For a comprehensive treatment of the X Toolkit, see Volume Four, *X Toolkit Intrinsics Programming Manual*, and Volume Five, *X Toolkit Intrinsics Reference Manual*.)

In Chapter 9, we'll take a look at some of the features common to applications written with the Motif Toolkit.

Redrawing the Screen: xrefresh

Depending on your environment, system messages may sometimes be displayed over the windows on your screen, obscuring them and making it difficult, if not impossible, for you to work. As described in Chapter 4, *More about the mwm Window Manager*, you can clear messages from the screen using the Refresh item on *mwm*'s Root Menu. The Refresh menu item actually runs a client called *xrefresh*, which redraws the screen as it was before the messages covered it.

When your screen is obscured, it's probably easiest to use the Refresh menu item, but you could also run *xrefresh* directly. Just focus input on an *xterm* window whose command line isn't obscured and enter:

```
% xrefresh
```

Then press Return. (You can run *xrefresh* in the foreground because it does its work and exits immediately.)

If the Refresh menu item is not available and you don't have a clear *xterm* to type in, you still may have a somewhat less desirable (but not entirely obvious) alternative: enter *xrefresh* in one of the obscured windows. Although it isn't readily apparent, you can actually enter text in any *xterm*, regardless of whether the window is obscured, if the window has the input focus. Thus, you can probably manage to click on (or place the pointer on) a window and peck out the *xrefresh* command on the keyboard, even if you're working blind.

You can avoid the problem of an obscured screen by running the *xconsole* client. *xconsole* provides a window to which system console messages are directed. For more information, see Appendix A, *Managing Your Environment*.

Desk Accessories

The clients *xclock*, *oclock*, *xcalc*, *xload*, *xbiff*, and *xman* can be thought of as *desk accessories*. (Desk accessories is a term we've borrowed from the Macintosh environment, meaning small applications that are available—and useful—at any time.)

You can start these clients from the command line in any *xterm* window or, if you like, you can add them to an *mwm* menu. (See Chapter 13, *Customizing mwm.*)

Clock Programs: xclock and oclock

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

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

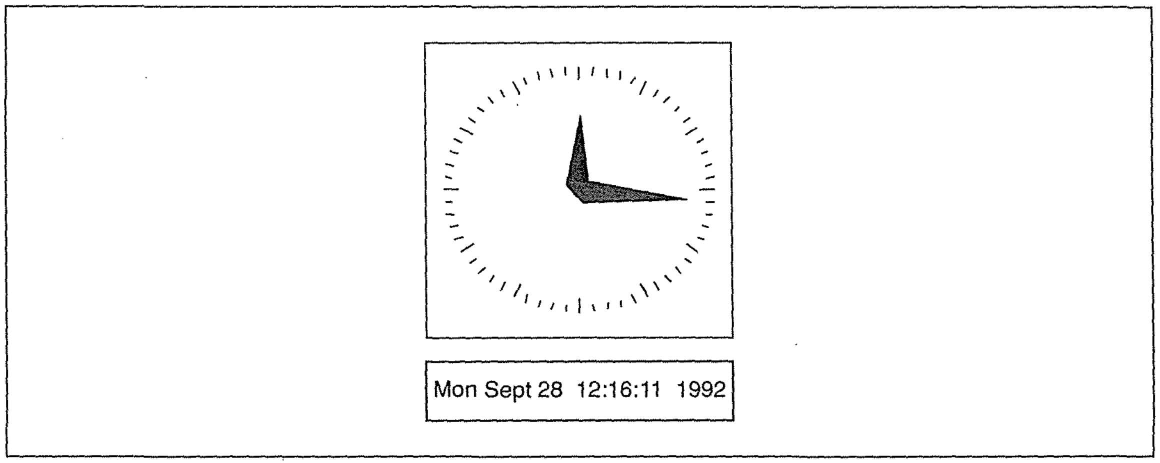


Figure 8-1. Two xclock displays: analog clock above digital clock

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

oclock also makes use of the X Shape Extension, which supports non-rectangular windows. If you try to resize the round *oclock*, you'll discover that it's possible to "stretch" it into various oblong shapes, as shown in Figure 8-2.

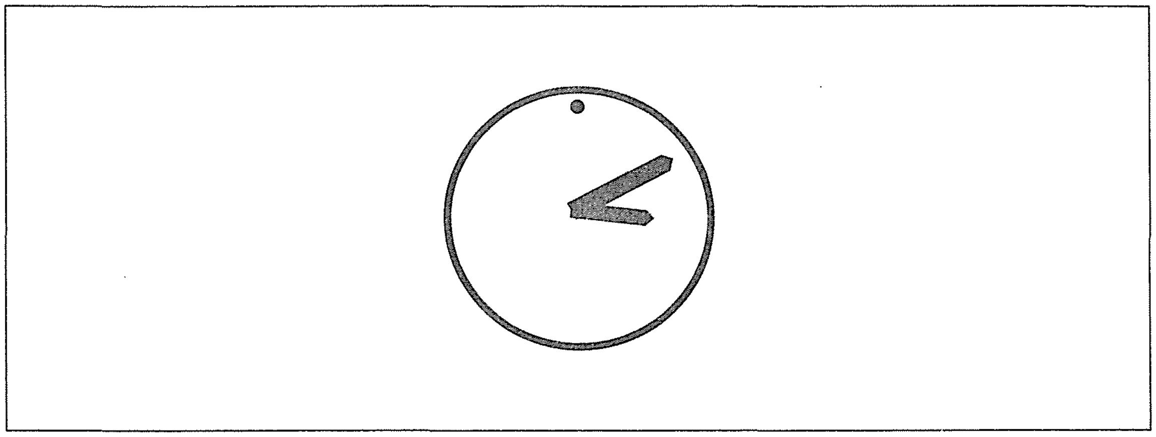


Figure 8-2. oclock display

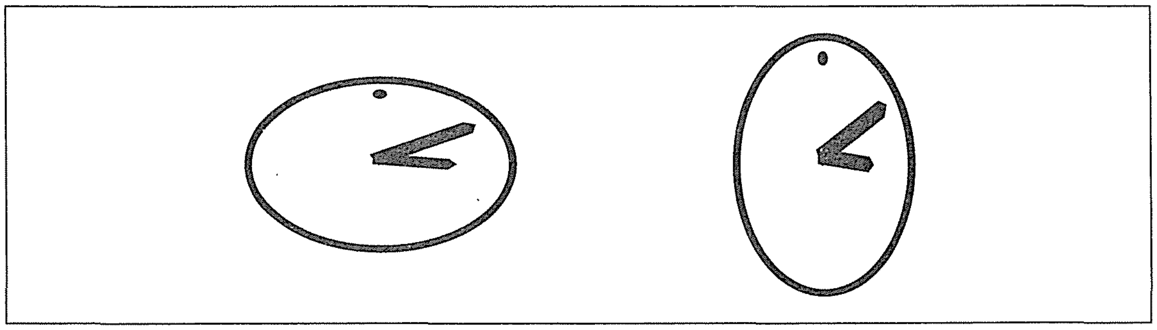


Figure 8-3. Oblong oclock displays

Note, however, that resizing is a little more complicated with *oclock* than with most other clients. When you run an application that creates a non-rectangular window with the 1.2 version of the *mwm* window manager, only a titlebar is displayed. The rest of the frame is suppressed, as shown in Figure 8-4.

Obviously, having only a titlebar limits how you can perform certain window manager functions. You can pull down the Window Menu and use the Minimize and Maximize buttons. Since you can't resize using the pointer on the frame, you either have to choose Size from the Window Menu or use the keyboard accelerator for the command (Alt+F8). Resizing from the rounded borders might also take some practice.

Having a titlebar with *oclock* has its advantages in window management, but you may not find it very aesthetic. See Chapter 13 for instructions on suppressing parts of the *mwm* window decoration.

If you don't suppress the window decoration, be aware that the titlebar may appear to blink on the minute when the clock is updated (redrawn).

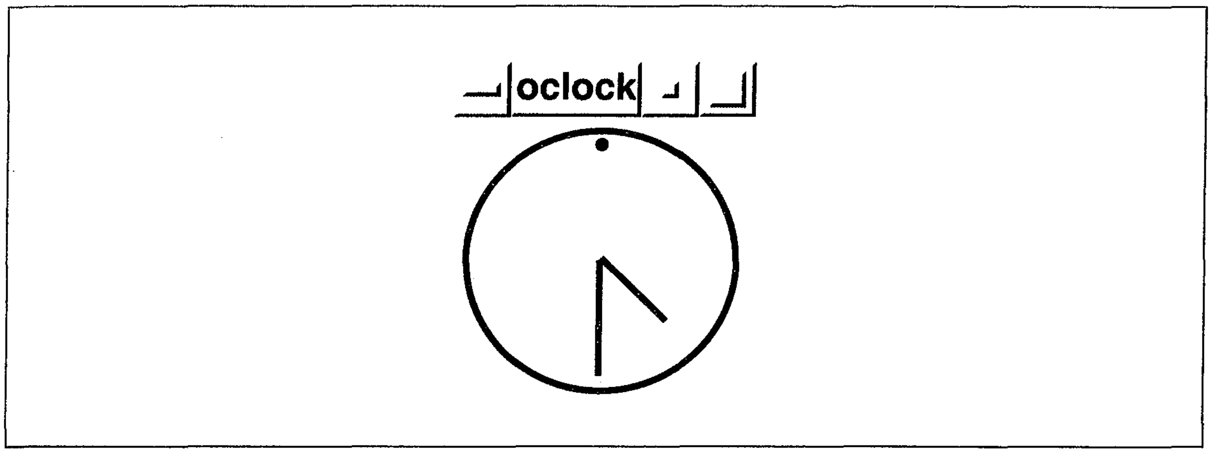


Figure 8-4. oclock with mwm titlebar

Running *oclock* with the `-transparent` option creates a transparent clock display, which can be fun, particularly on a root window with an interesting pattern. (You use the *xsetroot* client to specify root window characteristics. See Chapter 14 for details.)

Running a transparent *oclock* has a minor and not immediately obvious implication for window management. If you need to place the pointer on the transparent *oclock* "window," you must place it directly on the hands, border, or jewel—or on the titlebar, if it's displayed. When you place the pointer on the clock face, you're really pointing at whatever the background is—perhaps the root window, perhaps another application window. Since the *oclock* is transparent, it actually has no face!

This fact becomes important if you want to manage the *oclock* window using *mwm*. For instance, in many environments you can raise a window by clicking on any part of it. In this case, you'd be limited to the clock outline and the titlebar, if any.

You can also specify a color *oclock*. Though the default colors for *oclock* are black and white, it was designed to be run in color. The minute hand, hour hand, jewel, clock border, and background can all be set to a color. As of Release 5, the standard distribution of X includes a file of suggested color defaults for *oclock*. See Chapter 11, *Setting Resources*, and the *oclock* reference page in Part Three of this guide for instructions on using the suggested colors.

You can set your own color preferences (independent of the defaults) using either command-line options or by specifying client resources. See the *oclock* reference page in Part Three of this guide for the specific command-line options and resource variables.

Removing an xclock or oclock

Usually when you invoke *xclock* or *oclock* you will leave the clock running. However, if you experiment with these programs to test size, location, or color, you will notice that there is no obvious way to delete an unwanted clock. (Moving the cursor to the clock and pressing Control-C, Control-D, q, or Q doesn't work with *xclock* or *oclock*.)

You can remove an *xclock* or *oclock* window by using the *xkill* client, described later in this chapter. (Note that if you want to remove a transparent *oclock* with *xkill*, you have to click on the border, hands, jewel, or titlebar, if any.) You can also remove either clock by double-clicking on *mwm*'s Window Menu command button or by using the menu's Close item.

Another way to remove an *xclock* or *oclock* window is to identify and kill the process using the standard UNIX control mechanisms. First, find the process identification (PID) number for the client. For example, to determine the process ID number for *xclock*, go to an *xterm* window and type:

```
% ps -aux | grep xclock
```

at a system prompt. Under System V, type:

```
% ps -e | grep xclock
```

at a system prompt. The resulting display should look something like this:

```
128  p0  0:00  xclock
142  p0  0:00  grep xclock
```

The number in the first column is the process ID. Type:

```
% kill process_ID
```

The *xclock* display will be removed, and you will get the message:

```
Terminated      xclock
```

The same sequence of commands can be used to remove an *oclock* window.

Be aware, however, that these and other methods of "killing" a client have certain liabilities. See "Problems with Killing a Client" later in this chapter.

A Scientific Calculator: xcalc

xcalc is a scientific calculator that can emulate a Texas Instruments TI-30 or a Hewlett Packard HP-10C. Once you place the pointer within the *xcalc* window, the calculator can be operated in two ways:

- With the pointer, by clicking the first pointer button on the buttons in the calculator window.
- With the keyboard, by typing the same numbers and symbols that are displayed in the calculator window. (Most of the calculator keys have keyboard equivalents. The not-so-obvious equivalents are described on the *xcalc* reference page in Part Three of this guide.)

When using the first method, notice that the pointer appears as a small hand, enabling you to "press" the buttons. The values punched on the calculator and the results of the calculations are displayed in the long horizontal window along the top of the *xcalc*. Figure 8-5 shows *xcalc* on the screen.

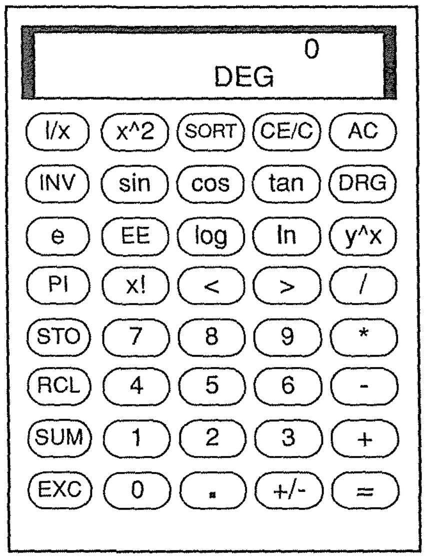


Figure 8-5. The default xcalc (TI-30 mode) on the screen

Figure 8-5 depicts the version of the calculator provided with Release 5 of X. As you can see, it features oval buttons. If you are running an earlier release, the calculator will have rectangular buttons and may also have a darker background color. These differences do not affect functionality. For additional information, see the *xcalc* reference page in Part Three of this guide.

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

```
% xcalc &
```

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

```
% xcalc -rpn &
```

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

xcalc allows you to select the number in the calculator display. You select the number using the first pointer button and paste it in another window using the second button. See Chapter 5, *The xterm Terminal Emulator*, for information about copying and pasting text selections. For more information on the function of each of the calculator keys, see the *xcalc* reference page in Part Three of this guide.

Terminating the Calculator

Terminate the calculator by either:

- Clicking the third pointer button on the TI calculator's AC key or the HP calculator's ON key, or:
- Positioning the pointer on the calculator and typing q, Q, or Control-C.

Mail Notification Client: xbiff

xbiff is a simple program that notifies you when you have mail. It puts up a window showing a picture of a mailbox. When you receive new mail, the keyboard emits a beep, the flag on the mailbox goes up, and the image changes to reverse video. Figure 8-6 shows the *xbiff* mailbox before and after mail is received.

After you read your mail, the image changes back to its original state. (You only need to run a mail program to convince *xbiff* that you've read your mail. You don't have to delete all messages.) Or, rather than run a mail program, you can simply click on the full mailbox icon with any pointer button to change it back to empty.

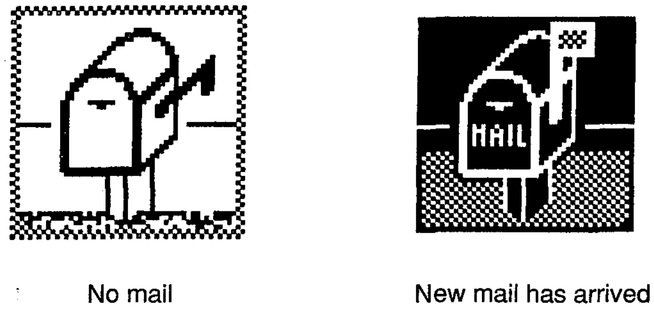


Figure 8-6. xbiff before and after mail is received

Regardless of whether you reset the mailbox to empty, *xbiff* causes the keyboard to beep each time more mail arrives.

The Release 5 version of *xbiff* is noticeably smarter than the R4 version which notified you of *any* change in the size of the mail file. (Thus, you were notified when you deleted messages and the file became smaller!)

While *xbiff* is intended to monitor a mail file, it can actually be set up to watch any file whose size changes using the `-file` option (followed by the name of the file to be monitored). For instance, if you're running a program that produces output intermittently, you can start *xbiff* with `-file` followed by the name of the output file; then *xbiff* will notify you when output is returned. (You can even specify an image other than the mailbox using resource variables—even for a single *xbiff* process.) See the *xbiff* reference page in Part Three of this guide for a list of options and resources. See Chapter 11, *Setting Resources*, for the syntax of resource specifications.

Monitoring System Load Average: xload

xload periodically polls the system for the load average and graphically displays that load using a simple histogram. By default, *xload* polls the system every ten seconds. You can change this frequency with the `-update` option. For example, if you enter this command in an *xterm* window:

```
% xload -update 3 &
```

the resulting *xload* window will poll the system every three seconds, as in Figure 8-7.

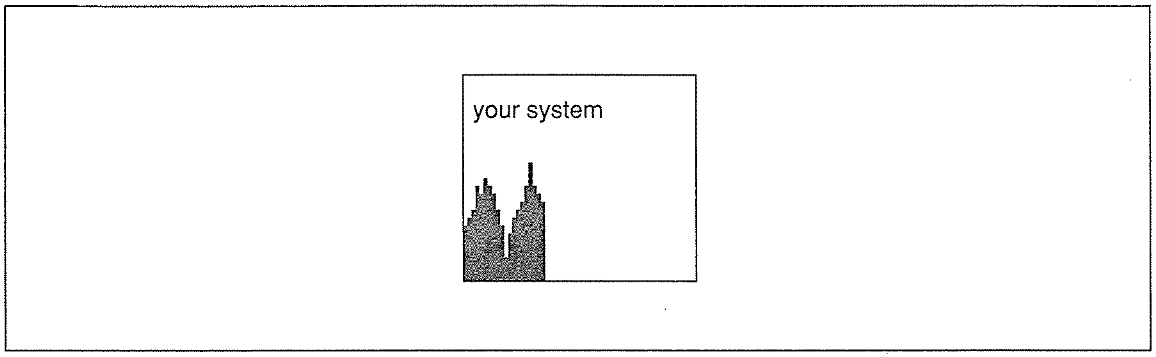


Figure 8-7. A sample xload window

If you are using both the local machine and remote machines, you can display loads for all systems and do your processing on the system that is fastest at the time.

Like *xclock* and *oclock*, *xload* provides no exit command. To remove an *xload* window, you need to kill the client process. See "Removing an xclock or oclock" earlier in this chapter.

Browsing Reference Pages: xman

The *xman* client allows you to display and browse through formatted versions of reference pages. By default, *xman* searches for manpages found in the directories specified by the MANPATH environment variable. (If you set the MANPATH, individual directory names should be separated by colons.) If MANPATH is not defined, *xman* searches the subdirectories of the directory */usr/man*. In a typical UNIX environment, there are 10 subdirectories: *man1* through *man8*, corresponding to the eight sections of reference pages in the UNIX documentation set; *manl* (man local) and *mann* (man new).

You run *xman* by typing:

```
% xman &
```

in an *xterm* window.

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

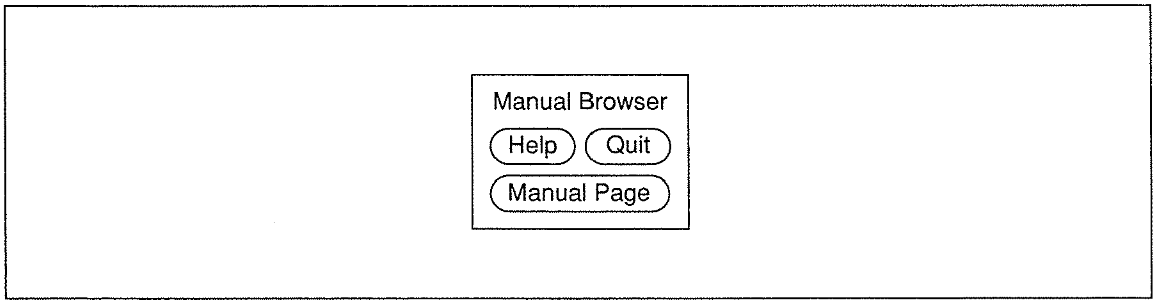


Figure 8-8. Initial xman window

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

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

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

If you place the pointer on the Options box and press and hold down the first button, a menu called Xman Options will be displayed below. The menu is pictured in Figure 8-9.

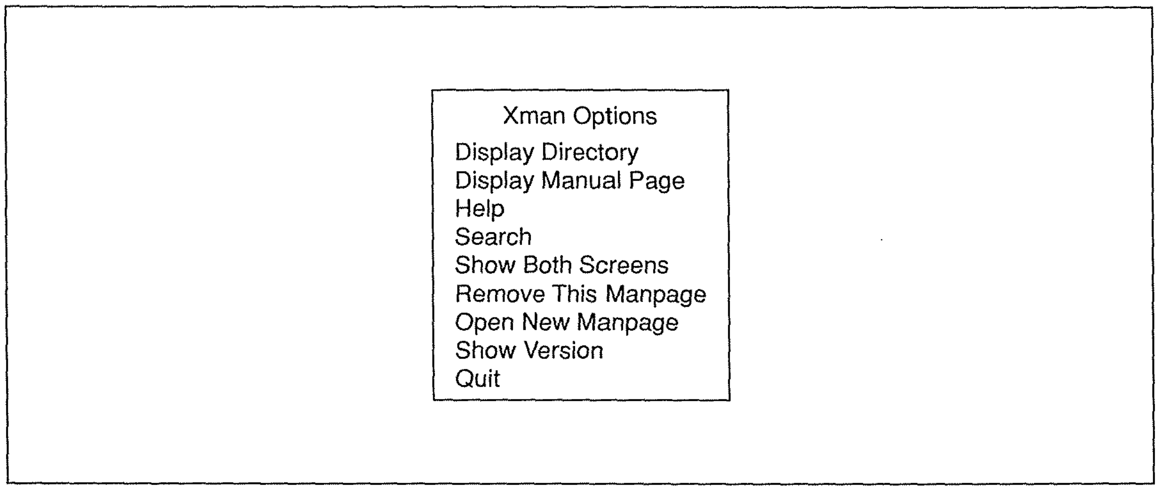


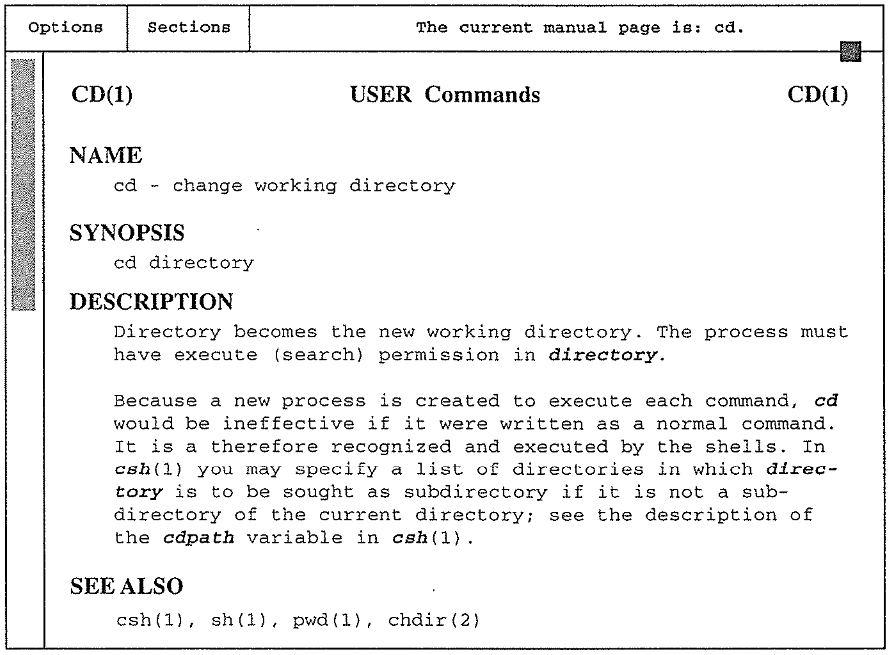
Figure 8-9. Xman Options menu

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

Display Directory lists the reference pages in the current reference page directory (also called a *section*). By default, this is *man1*, the user commands. When you list the contents of *man1* in this way, the informational section of the horizontal bar reads Directory of: (1) User Commands.

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

Once you've listed a reference page directory in the *xman* window, you can display a formatted version of any page in the list simply by clicking on the name with the first pointer button. Figure 8-10 shows the formatted reference page for the UNIX *cd*(1) command.



```
CD(1)                    USER Commands                    CD(1)

NAME
     cd - change working directory

SYNOPSIS
     cd directory

DESCRIPTION
     Directory becomes the new working directory. The process must
     have execute (search) permission in directory.

     Because a new process is created to execute each command, cd
     would be ineffective if it were written as a normal command.
     It is a therefore recognized and executed by the shells. In
     csh(1) you may specify a list of directories in which direc-
     tory is to be sought as subdirectory if it is not a sub-
     directory of the current directory; see the description of
     the cdpath variable in csh(1).

SEE ALSO
     csh(1), sh(1), pwd(1), chdir(2)
```

Figure 8-10. cd reference page displayed in xman window

As we'll see later in this section, if you know the name of the page you want to display, you can skip displaying the relevant section listing and simply search for the name. For now, let's assume you need to glance at the directory listings to find what you want.

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

To display a manual page from another directory in the MANPATH, you can change to that directory using the Xman Sections menu. Bring up the menu by placing the pointer in the Sections box in the application's titlebar and holding down the first button. The Xman Sections menu lists the directories of UNIX manual pages under the MANPATH, as shown in Figure 8-11.

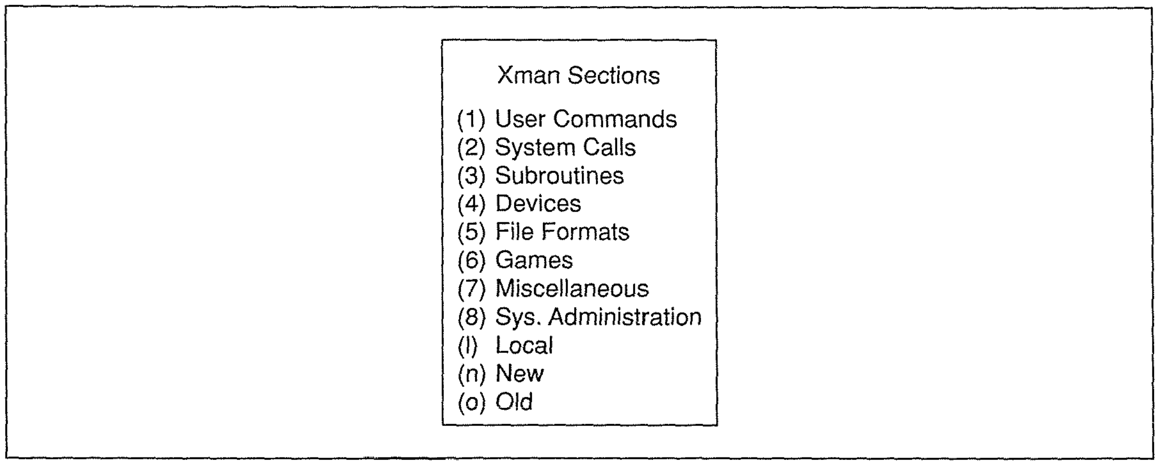


Figure 8-11. Xman Sections menu

Keep in mind that the available sections may vary from system to system. Figure 8-11 shows the manual page sections typical to a UNIX environment.

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

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

If you know the name of the page you want to display (regardless of the section), you don't need to use either of *xman*'s menus. While the *xman* window has the input focus, typing Control-S pops up a dialog box prompting for a specific manual page to display, as in Figure 8-12.

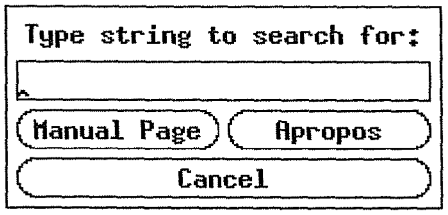


Figure 8-12. xman's search dialog prompts for a manpage to display

Note that the dialog box takes over the input focus. If you know the exact name of the page you want, type it and either press Return or click on the Manual Page button in the search window. The specific page will be displayed in a browsing window. (If you requested the search from an existing Manual Page window, the new page will be displayed there. If you requested the search from the initial *xman* window, the page will be displayed in a new browsing window.)

If you don't know the exact name of the manpage you want, you can type a guess in the search window and click on the Apropos button, which displays a list of pages containing the string you've entered. Note, however, that you cannot select a name from the Apropos list. Instead, type Control-S again and search for the exact name you want.

You can iconify any of the various windows *xman* creates; each one will be represented by a separate icon symbol.

You can remove a browsing window by selecting the Remove This Manpage option from the Xman Options menu.

Selecting Quit from the Xman Options menu or from the initial *xman* window causes the client to exit.

xman displays each manpage directory in a window known as a *viewport*, created with the Athena Viewport widget from the X Toolkit. A viewport is a composite widget that provides a main window and horizontal and/or vertical scrollbars. The Athena Viewport widget is analogous to the Motif Toolkit's ScrolledWindow, described in Chapter 9, *Working with Motif Applications*.

xman's scrollbar is also an Athena widget. (See Chapter 5, *The xterm Terminal Emulator*, for instructions on using an Athena scrollbar.) The Motif Toolkit also provides a scrollbar widget, described in Chapter 9, which looks and operates somewhat differently than the Athena scrollbar.

The xedit Text Editor

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

The *xedit* client is valuable to illustrate the use of the Athena Text widget. (*xedit* can also be used to illustrate several other widgets.) However, we do not recommend using *xedit* as your primary text editor. The program is somewhat buggy and its behavior can be erratic. For example, it's fairly easy to overwrite files inadvertently, as explained in the discussion of the Load button later in this section. The redraw command (Control-l) causes text in the window to scroll so as to reposition the cursor in the center of the editing window—not a welcome surprise. Some of the commands to create a new paragraph may also inadvertently copy preceding text. These are just a few of *xedit*'s inconvenient features.

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

xedit recognizes various Control and Meta keystroke combinations that are bound to a set of commands similar to those provided by the *emacs* text editor.* In addition, you can use the

*The commands may be bound to keys different from the defaults described below through the standard X Toolkit key translation mechanisms. See Chapter 11, *Setting Resources*, for more information.

pointer to move the cursor in the text or to select a portion of text. The *xedit* cursor is a caret symbol (^). A caret cursor appears in each of the three areas that accept text entry. (These areas are described later in this section.) Pressing the first pointer button causes the insertion point (cursor) to move to the location of the pointer. Notice that the cursor always appears between characters, rather than on a character as the *xterm* cursor does. Double-clicking the first pointer button selects a word, triple-clicking selects a paragraph, and quadruple-clicking selects everything. After you select text, the selection may be extended in either direction by using the third pointer button.

You can invoke *xedit* by entering:

```
    % xedit &
```

Since no filename has been specified, the main section of the *xedit* window is empty, as illustrated by Figure 8-13.

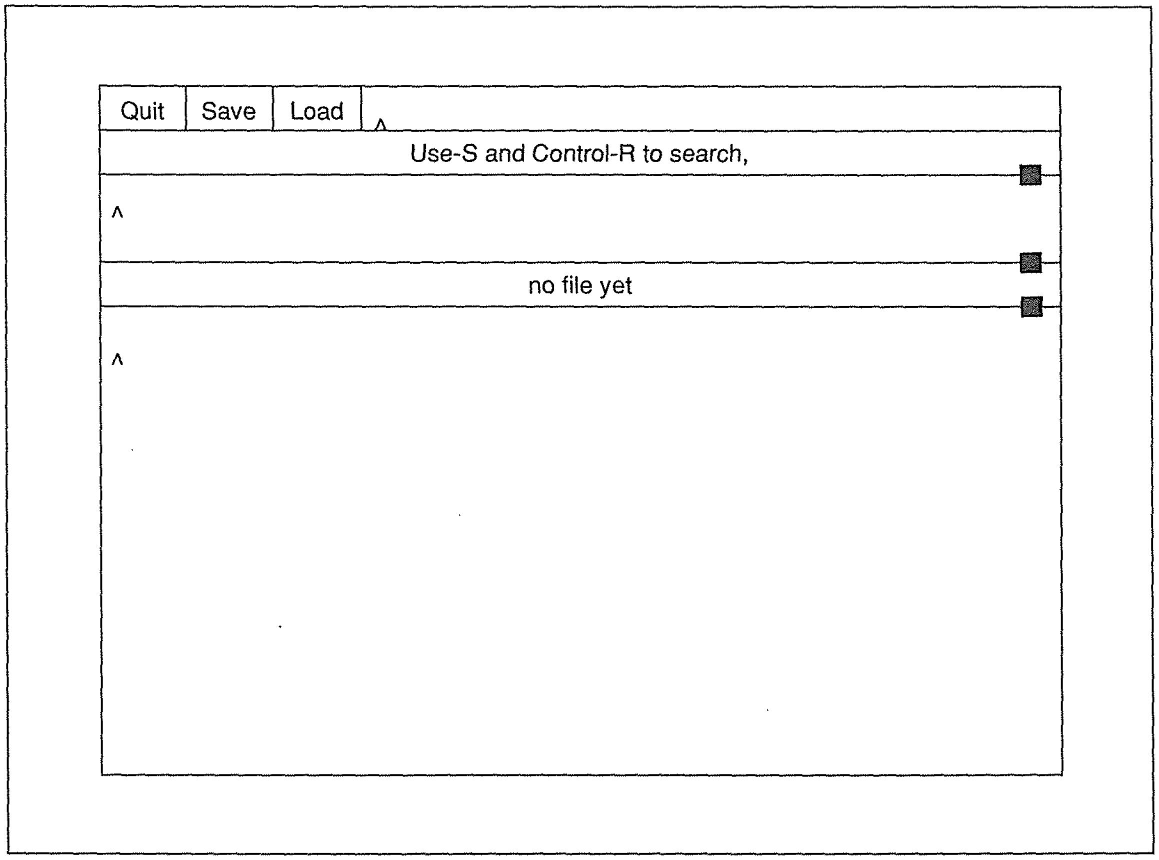


Figure 8-13. xedit window before text file is read in

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

- A commands section, which features three command push buttons (Quit, Save, and Load) and an area to their right in which a filename can be entered.

- A message window, which displays messages from the client and can also be used as a scratch pad.
- The filename display, which shows the name of the file being edited and the read/write permissions for the file.
- The edit window, in which the text of the file is displayed and in which you issue the editing commands.

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

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

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

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

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

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

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

Save	Writes the file. If file backups are enabled (using the `enableBackups` resource), *xedit* first stores a copy of the unedited file as ***filename**.BAK* and then overwrites *filename* with the contents of the edit window. The *filename* used is the text that appears in the area immediately to the right of the Load button.
Load	Loads the file displayed immediately to the right of the button into the edit window. If a file is currently being displayed and has been modified, a warning message will ask the user to save the changes, or to press Load again.

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

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

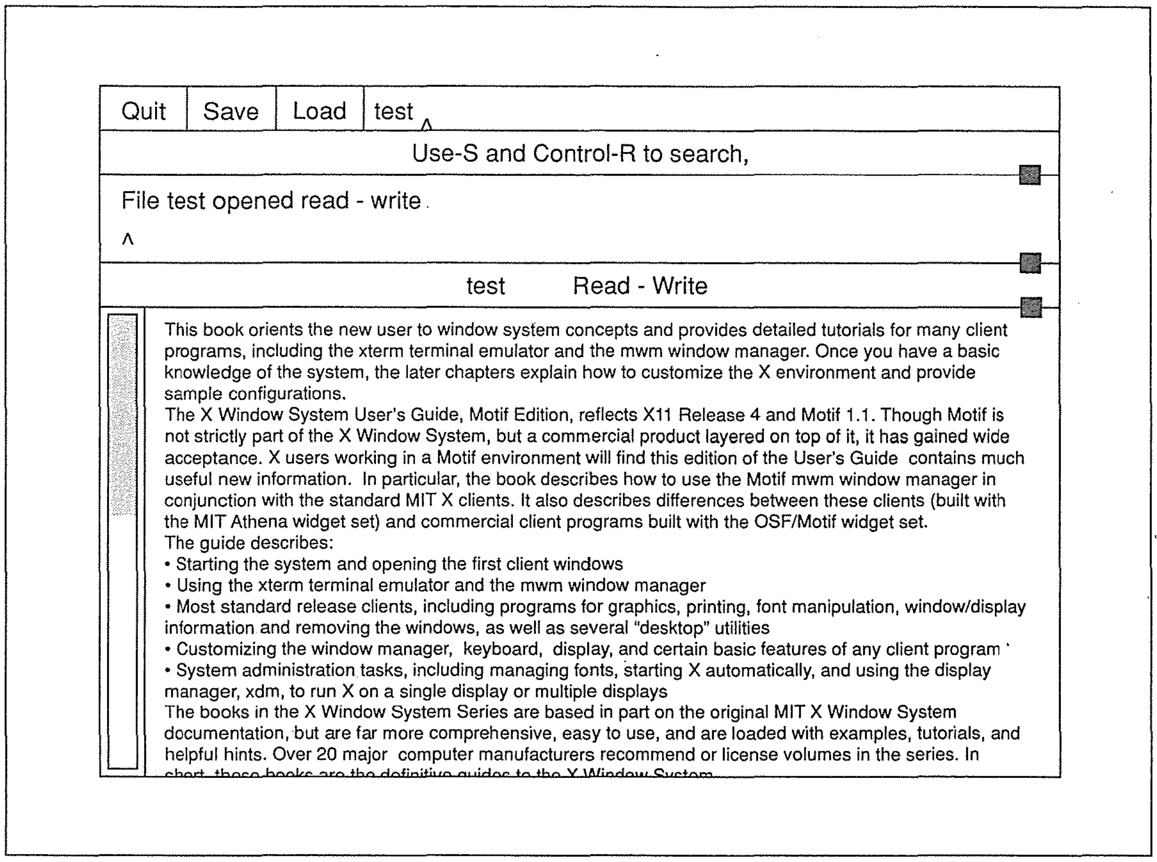
A second problem can occur after you've loaded a file by entering the name in the window next to the Load button. Say you've been editing the file for some time, but haven't saved the changes. If you go to save the changes and accidentally double-click on Load (not that difficult to do), you'll reload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

Now, after considering some of the possible pitfalls, let's load a file into the empty *xedit* window as shown in Figure 8-13. (Obviously, in this case, there's no danger of overwriting an existing file.) To load a file called *test*:

1. Place the pointer in the area to the right of the Load button.
2. Type *test*. Notice that the caret cursor moves as you type.
3. Place the pointer on the Load command button and press the first pointer button.

The file called *test* is displayed in the edit window, as shown in Figure 8-14.

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



File test opened read - write.

This book orients the new user to window system concepts and provides detailed tutorials for many client programs, including the xterm terminal emulator and the mwm window manager. Once you have a basic knowledge of the system, the later chapters explain how to customize the X environment and provide sample configurations.
The X Window System User's Guide, Motif Edition, reflects X11 Release 4 and Motif 1.1. Though Motif is not strictly part of the X Window System, but a commercial product layered on top of it, it has gained wide acceptance. X users working in a Motif environment will find this edition of the User's Guide contains much useful new information. In particular, the book describes how to use the Motif mwm window manager in conjunction with the standard MIT X clients. It also describes differences between these clients (built with the MIT Athena widget set) and commercial client programs built with the OSF/Motif widget set.
The guide describes:
• Starting the system and opening the first client windows
• Using the xterm terminal emulator and the mwm window manager
• Most standard release clients, including programs for graphics, printing, font manipulation, window/display information and removing the windows, as well as several "desktop" utilities
• Customizing the window manager, keyboard, display, and certain basic features of any client program
• System administration tasks, including managing fonts, starting X automatically, and using the display manager, xdm, to run X on a single display or multiple displays
The books in the X Window System Series are based in part on the original MIT X Window System documentation, but are far more comprehensive, easy to use, and are loaded with examples, tutorials, and helpful hints. Over 20 major computer manufacturers recommend or license volumes in the series. In short, these books are the definitive guides to the X Window System.

Figure 8-14. test file displayed in edit window

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

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

Keep in mind that you can redefine any of these key combinations using what are known as *translations*. Translations allow you to assign actions recognized by a client to particular key combinations, or key and pointer button combinations. For example, *xedit* recognizes actions to delete text, to copy text, to move the cursor, etc. *xedit* defines key combinations to invoke these actions. (The key/action mappings appear in the list at the end of this section.) For information on specifying alternate mappings, see "Event Translations" in Chapter 11, *Setting Resources*.

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

Control-a	Move to the beginning of the current line.
Control-b	Move backward one character.
Control-d	Delete the next character.
Control-e	Move to the end of the current line.
Control-f	Move forward one character.
Control-h or Backspace	Delete the previous character.
Control-j, Control-m, Return, LineFeed	New line.
Control-k	Kill the rest of this line. (Does not kill the carriage return at the end of the line. To do so, use Control-K twice. However, be aware that the second kill overwrites the text line in the kill buffer.)
Control-l	Redraw the window. (Also scrolls text so that cursor is positioned in the middle of the window.)
Control-n	Move down to the next line
Control-o	Divide this line into two lines at this point and move the cursor back up.
Control-p	Move up to the previous line.
Control-r	Search and replace backward.
Control-s	Search and replace forward.
Control-t	Transpose characters. (Swap the characters immediately before and after the cursor.)
Control-u	Control-u, Control-n moves the cursor down four lines.
Control-v	Move down to the next screenful of text.
Control-w	Kill the selected text.
Control-y	Insert the last killed text. (If the last killed text is a carriage return—see Control-k above—a blank line is inserted.)
Control-z	Scroll up the text one line.

Meta-<	Move to the beginning of the file.
Meta->	Move to the end of the file.
Meta-[	Move backward one paragraph.
Meta-]	Move forward one paragraph.
Meta-b	Move backward one word.
Meta-d	Delete the next word.
Meta-D	Kill the next word.
Meta-f	Move forward one word.
Meta-h, Meta-Backspace, Meta-Delete	Delete the previous word.
Meta-H, Meta-Shift-Backspace, Meta-Shift-Delete	Kill the previous word.
Meta-i	Insert a file. A dialog box will appear in which you can type the desired filename.
Meta-k	Kill to the end of the paragraph.
Meta-q	Join lines to form a paragraph.
Meta-v	Move up to the previous screenful of text.
Meta-y	Insert the last selected text here. This command is the equivalent of clicking the second pointer button. See Chapter 5, *The xterm Terminal Emulator*, for more information about text selections.
Meta-z	Scroll down the text one line.
Delete	Delete the previous character.

Printing Utilities: xwd, xpr, xdpr

xwd stores window images in a formatted window dump file. This file can be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing, etc.

To create a window dump file, type:

```
% xwd > file
```

The pointer will change to a small crosshair symbol. Move the crosshair pointer to the desired window and click any button. The keyboard bell rings once when the dump starts and twice in rapid succession when the dump is finished.*

To make a dump of the entire root window (and all windows on it), use the `-root` option:

```
% xwd -root > file
```

When you select a single window, by default *xwd* takes an image of the window proper. To include a window manager frame or titlebar, use the `-frame` option.

xwd allows you to capture a single window or the entire root window. But what if you want an image that includes more than one window or parts of multiple windows? You can use *xmag* (described in Chapter 7) to capture an image of multiple windows and then use *xwd* on the *xmag* window! Since *xmag* is intended to magnify, if you want the window image to be the actual size, you must specify that no magnification is performed. To do this, you run *xmag* with the option `-mag 1`. See Chapter 7, *Graphics Utilities*, and the *xmag* reference page in Part Three of this guide for more information.

To redisplay a file created with *xwd* in a window on the screen, use the *xwud* client, an undumping utility. Specify the dump file to display as an argument to the `-in` option:

```
% xwud -in file
```

Then remove the image by typing Control-C in the *xterm* from which you started *xwud*.

xpr takes as input an X Window System dump file produced by *xwd* and converts it to a printer-specific format that can be printed on a PostScript printer (such as the Apple LaserWriter), the Digital LN03 or LA100 printer, the IBM PP3812 page printer, the HP LaserJet (or other PCL printers) or the HP PaintJet. By default, output is formatted for PostScript. Use the `-device` option to format for another printer. For example, to format a window dump file for the DEC LN03 printer, type:

```
% xpr -device ln03 file > file.ps
```

Other options allow you to change the size, add headers or footers, and so on. See the *xpr* reference page in Part Three of this guide for details.

You can use *xwd* and *xpr* together, using the standard UNIX pipe mechanism. For example:

```
% xwd | xpr -device ln03 | lp
```

*If the keyboard bell is turned off, you won't hear the double beep that signals the dump is finished. You control bell volume with the *xset* client, described in Chapter 14, *Setup Clients*.

The *xdpr* command combines these three separate commands into one. (On System V, *lp* is used; on BSD-based systems, *xdpr* uses *lpr*.) *xdpr* accepts most of the options accepted by *xwd*, *xpr*, and *lp*(1) or *lpr*(1). Thus, you could use the command:

```
% xdpr
```

to take a window dump (xwd), convert that file to PostScript (`xpr` converts to PostScript by default), and print the output (*lpr* or *lp*). See the *xdpr* reference page in Part Three of this guide for more information.

Note that when you start piping together the output of X clients, you run into some ambiguities. For example, if you pipe the output of *xwd* to *xpr* and for some reason the *xpr* command fails, *xwd* will still be there waiting for pointer input. The original UNIX pipe mechanism doesn't have the concept of data dependent on pointer input! The integration of the UNIX model of computing (in which standard input and output are always recognized) and the window model is not always complete, sometimes leading to unexpected behavior.

As an even more flagrant example, you can create a pipe between two programs, the first of which doesn't produce standard output and the second of which doesn't recognize standard input. The shell doesn't know any better and the programs themselves go on their merry way with pointer and windows.

However, it is nice to know that you can pipe together program output, even when some of those programs may not produce output until you intervene with the pointer.

Even without pipes, you should start thinking about how these programs could work together. For example, you could create and print a picture of a font using the following steps:

1. Display a font with *xfd*. (See Chapter 6, *Font Specification*, for instructions on how to use *xfd*.)
2. Resize the window to improve readability, using the *mwm* resize handles or the Resize item of the Window Menu.
3. Create a window dump file with the command `xwd > file`.
4. Create a PostScript file from the dump with the command:

   ```
   xpr file> file.ps
   ```

5. Print the PostScript file the appropriate print command (*lp* or *lpr*).

Even though the UNIX shell will accept a pipe between *xfd*, *xwd*, and *xpr*, what actually happens is that *xwd* starts up faster than *xfd* and is ready to dump a window before the *xfd* window appears.

Killing a Client Window with xkill

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

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

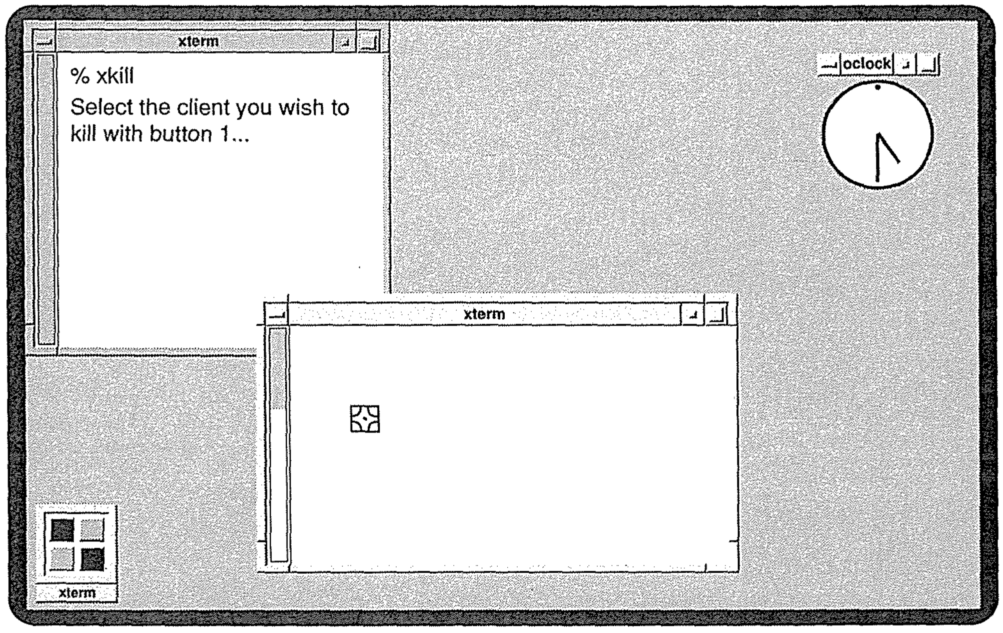
xkill is intended primarily to be used in cases where more conventional methods of removing a client window do not work. It is especially useful when programs have displayed undesired windows on the screen. To remove a stubborn client window, type:

```
    % xkill
```

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

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

Move the draped box pointer to the window you want to remove, as shown in Figure 8-15, and click the first pointer button. The window is removed. (*xkill* does not allow you to select the root window.)



```
% xkill
Select the client you wish to
kill with button 1...
```

Figure 8-15. Selecting the window to be removed

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

To remove a window using its ID number, type:

```
% xkill -id number
```

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

Problems with Killing a Client

The most obvious problem with *xkill* is that it's possible to kill the wrong window inadvertently. Perhaps less obvious is a problem inherent in "killing" a program. As a general rule, a command that "kills" a program does not give the program time to save or complete processes that are still running—in effect, to clean up after itself. The processes that can be adversely affected may be visible to the user, such as an editing session, or they may be underlying system processes, such as writing to or reading from a socket.

Most clients can be terminated in ways that allow them to finish all relevant processes and then exit cleanly. These methods should be attempted *before* you use *xkill* or some other method that kills the client.

For example, you can generally remove an *xterm* window by typing in the window the same command you use to log off the system. You should also be able to remove an *xterm* window with various Main Options menu commands, depending on the signals that can be interpreted by your system. (Some of these signals, such as SIGHUP and SIGTERM, are more gentle to the system. See the *xterm* reference page in Part Three of this guide for a list of menu commands and the signals they send.) Both *xcalc* and *bitmap* can be removed by typing q, Q, or Control-C in the window. The *bitmap* File menu also has a Quit item.

A few clients, such as *oclock*, cannot be removed *except* by killing. You must use *xkill* or a similar method to remove an *oclock* window.

Generally, however, you should exhaust the safer alternatives before you use *xkill* and other commands that kill a client.

When you want to remove a window, depending on the client and what commands it recognizes, try these methods (roughly) in the following order.

1. Methods that cause the client to exit after finishing relevant processes:

    a. Special commands (e.g., `exit`) or key sequences (e.g., Control-D, Control-C, q, Q) recommended to stop a client.

    b. Certain application-specific menu items (e.g., for *xterm*, the Main Options menu commands Send HUP Signal, Send TERM Signal, and Quit; the Quit item on the *bitmap* File menu).

2. When these methods don't work or don't apply (as in the case of *oclock*), *then* use commands or menu items that kill the client:

    a. The Close item on the *mwm* Window Menu; or a double click on the Window Menu command button.

    b. The *xkill* client.

    c. The UNIX *kill* command with the client's process ID number, which is determined using *ps*. (This method of removing a window is described for *xclock* and *oclock* earlier in this chapter.*)

    d. For removing *xterm* windows only: the Send KILL Signal item on the client's Main Options menu.

Window and Display Information Clients

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

Displaying Information About a Window: xwininfo

The *xwininfo* client can display information about a particular window or about how windows are related to one another on the display. (We'll discuss this latter capability in the next section.) Among the most useful information the client provides is the size and location of the window—in the form of a geometry string that can be specified on the command line. (Chapter 3 describes how to specify a window's size and location using the `-geometry` option.)

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

xwininfo also provides you with the *window ID* (also called the resource ID). Each window has a unique identification number associated with it. This number can be used as a command-line argument with several clients. Most notably, the window ID can be supplied to the *xkill* client to specify the window be killed.

You can also use the window ID as an argument to the *xprop* client, which displays various window properties. As described in Chapter 1, a property is a piece of information associated with a window or a font and stored in the server. Properties facilitate interclient communication; they are used by clients to store information that other clients might need to know. Storing properties in the server makes the information they contain accessible to all clients.

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

```
% xwininfo
```

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

```
xwininfo: Please select the window about which you
          would like information by clicking the
          mouse in that window.
```

You can select any window on the display, including the window in which you've typed the command and the root window. (Rather than using the pointer, you can specify a window on the command line by supplying its title, or name if it has no title, as an argument to *xwininfo*'s own -name option. See Chapter 10 for information about setting a client's title and name. See the *xwininfo* reference page in Part Three of this guide for a list of its options.)

Example 8-1 shows the statistics *xwininfo* supplies, with some typical readings.

Example 8-1. Window information displayed by xwininfo

```
xwininfo: Window id: 0x280000f "xterm"

  Absolute upper-left X:  8
  Absolute upper-left Y:  25
  Relative upper-left X:  0
  Relative upper-left Y:  0
  Width: 819
  Height: 484
  Depth: 8
  Visual Class: PseudoColor
  Border width: 0
  Class: InputOutput
  Colormap: 0x27 (installed)
  Bit Gravity State: NorthWestGravity
  Window Gravity State: NorthWestGravity
  Backing Store State: NotUseful
  Save Under State: no
  Map State: IsViewable
  Override Redirect State: no
  Corners:  +8+25  -325+25  -325-391  +8-391
  -geometry 80x24+0+0
```

These readings are for a login *xterm* window displayed using a 10 × 20 pixel fixed-width font. The *mwm* window manager is also running. Most of the numerical information is in pixels. Depth is in bits per pixel. The colormap is represented by a hexadecimal number. Since we've selected an *xterm* window, the width and height components of the geometry string are in characters and lines (80 × 24).

For the average user, the first and last lines from Example 8-1 are most significant:

```
    xwininfo: Window id: 0x280000f "xterm"
                              .
                              .
                              .
      -geometry 80x24+0+0
```

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

Probably most important is the final line, which provides the appropriate `-geometry` option to create this client window. (As described in Chapter 3, `-geometry` allows you to specify a window's size and position.)

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

```
      Absolute upper-left X:  8
      Absolute upper-left Y:  25
      Relative upper-left X:  0
      Relative upper-left Y:  0
      Width: 819
      Height: 484
      Depth: 8
      Visual Class: PseudoColor
      Border width: 0
      Class: InputOutput
      Colormap: 0x27 (installed)
      Bit Gravity State: NorthWestGravity
      Window Gravity State: NorthWestGravity
      Backing Store State: NotUseful
      Save Under State: no
      Map State: IsViewable
      Override Redirect State: no
      Corners:  +8+25  -325+25  -325-391  +8-391
```

Generally, the absolute upper-left X and Y correspond to the positive x and y offsets (the third and fourth components of the geometry string). However, the *mwm* frame complicates matters. When *mwm* is running, the absolute upper-left X and Y correspond to the x and y coordinates of the application window—but not the framed window!

Let's take another look at the sample *xwininfo* output. The absolute upper-left X and Y suggest that the window is located at coordinates 8,25. However, the output is actually for an *xterm* located at coordinates 0,0! The 8,25 are the coordinates of the *xterm* window itself; the coordinates represent the distance of the window from 0,0 *including the dimensions of the frame*. The default frame is actually 8 pixels in the x dimension and 25 pixels in the y dimension (because of the titlebar).

The relative upper-left X and Y are not meaningful if you're running *mwm*. Regardless of a window's location, the relative upper-left X and Y are 0 and 0.

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

The values for window depth and colormap relate to how color is specified. See Chapter 12, *Specifying Color*, for more information.

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

Examining the Window Hierarchy

Windows are arranged in a hierarchy, much like a family tree, with the root window at the top. Prior to Release 5, you could display the window tree (starting with the root window) using the *xlswins* client. In Release 5, *xlswins* is no longer available. Instead, you can perform the same functions using *xwininfo* with the `-children` and `-tree` options.

The `-children` and `-tree` options give information about how windows on the display are related to one another. Remember that most application windows are actually composed of several subwindows. These options also allow you to see a client's internal window hierarchy.

Naturally, the information returned using these options largely depends on the window you choose with the pointer. If you choose an application window (such as an *xterm*), you get information about that client's internal hierarchy. If you choose the root window, you get information about all of the application windows on the display—which are the children of the root window.

The command:

```
% xwininfo -children
```

gives information about the immediate children of the window you select (a single generation). Example 8-2 shows the results of this command for an *xterm* window at coordinates 0,0 on the root window. (No window manager is running.)

Example 8-2. Window tree displayed by xwininfo –children

```
xwininfo: Window id: 0x2c0000f "xterm"

  Root window id: 0x2d (the root window) (has no name)
  Parent window id: 0x2d (the root window) (has no name)
     1 child:
     0x2c00016 (has no name): ()  819x484+0+0  +1+1
```

Any client that displays an application window, such as *xterm*, *xclock*, *xfd*, *bitmap*, etc., will be listed by name (in quotes) following the window ID number.* The remaining lines show how the selected window fits in the hierarchy. First the root window is listed, with its ID number. (Obviously, this line will be the same, regardless of the window you select.) Client windows displayed on the root window are called *children* of the root window, in keeping with the family tree analogy; thus, the root window is also the parent of the *xterm* window. The "1 child" of the *xterm* application window is the *xterm* VT102 window. (The first line lists the application shell window, which can be displayed both as a VT102 and a Tektronix window.) In the *xwininfo* listing, a child window is indented once under its parent.

Notice the geometry information at the end of the final line. The first geometry string gives the dimensions of the VT102 window in pixels and its coordinates relative to the *root* window. Since *mwm* is not running, frames are not an issue. Thus, a window at coordinates 0,0 has the position `+0+0` relative to the root. The second geometry string gives the window's coordinates relative to its *parent* window. The VT102 window is `+1,+1` pixels from the application window because of a single pixel border.

The `-children` option displays a single "generation" below the selected window. If you run *xwininfo* with this option and select the root window, you'll get the resource ID numbers for all client windows on the display. As you might recall, you can supply this number to *xkill* to specify the window to kill. You can also supply a resource ID to *xwininfo* to specify the window you want information about, or to *xprop* to get the window's properties. Being able to display the ID numbers of all client windows on the screen simultaneously is especially helpful if one or more windows is obscured in the stack. Running `xwininfo -children` and selecting the root can help you determine by process of elimination which window is hidden—without having to circulate all the windows on your screen. You can then use the resource ID number of that window in the ways we've discussed.

To list additional generations of windows, run *xwininfo* with the `-tree` option. In Example 8-3, we've used `-tree` and then selected the root window.

Example 8-3. Run xwininfo –tree and select the root to see the complete window hierarchy

```
xwininfo: Window id: 0x2d (the root window) (has no name)

  Root window id: 0x2d (the root window) (has no name)
  Parent window id: 0x0 (none)
     2 children:
     0x300000a "xclock": ("xclock" "XClock")  164x164+986+0  +986+0
        1 child:
        0x300000b (has no name): ()  164x164+0+0  +987+1
     0x2c0000f "xterm": ("xterm" "XTerm")  819x484+0+0  +0+0
        1 child:
        0x2c00016 (has no name): ()  819x484+0+0  +1+1
           1 child:
           0x2c00017 (has no name): ()  14x484+-1+-1  +0+0
```

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

You'll probably notice the application windows "xclock" and "xterm" right away. These are listed as the "2 children" of the "Parent window," which in this case is the root window. But what are the other windows listed in Example 8-3? A superficial examination of these other windows provides a brief introduction to the inner workings of X. An underlying feature of X is that menus, boxes, icons, and even *features* of client windows, such as scrollbars, are actually windows in their own right. What's more, these windows (and client window icons) may still exist, even when they are not displayed.

The three remaining windows are unnamed. From the relative indents of the windows, we can tell certain information. The first unnamed window is a child of the *xclock*. From our prior examples, it seems clear that this is the client window running under the application shell. The second unnamed window is a child of the *xterm*. (As we've seen, this is the VT102 window) The third unnamed window is a child of the child (or the *grandchild* of the *xterm*). The geometry information can help us identify this window. It is the window's scrollbar.

An additional note about the `-tree` listing. On the line introducing each application window are what are known as the instance and class resource names for the client (for example, `xterm`, `XTerm`). You use the instance and class resource names to specify default window characteristics, generally by placing them in a file in your home directory. This is described in detail in Chapter 11, *Setting Resources*.

The listing in Example 8-3 was generated when the *mwm* window manager was not running. If *mwm* is running, the output is considerably more complicated. Many of the features provided by *mwm*, such as the window frame and its command buttons, and the Root Menu and Window Menu, are actually all windows. This *greatly* complicates the window hierarchy. If you run *xwininfo –tree* while *mwm* is running and select the root, even if the display has only a single application window, the output will be dozens of lines long; you can assume that most of the mysterious windows in the hierarchy are features provided by the window manager.

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

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

Listing the Currently Running Clients: xlsclients

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

```
harry  xterm -geometry 80x24+10+10 -ls
harry  xclock -geometry -0+0
ruby   xterm -geometry 80x24-0-0 -display harry:0.0
```

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

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

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

Note, however, that the *xlsclients* listing has limitations. Our sample output above, for instance, does not explain how the remote process (on ruby) was started. From this information, it's impossible to tell if the user logged on to ruby (perhaps using *rlogin* or *telnet*) and started an *xterm* or simply ran a remote shell (*rsh*) from the machine harry. In such cases, you have to edit *xlsclients* output in order to incorporate it into a login script. From the previous output, we might come up with the script:

```
xclock -geometry -0+0 &
rsh ruby xterm -geometry 80x24-0-0 -display harry:0.0 &
xterm -geometry 80x24+10+10 -ls
```

(Because of the way X works, the final process must remain in the foreground. See Appendix A, *Managing Your Environment*, for an explanation.)

Keep in mind, also, that *xlsclients* only lists clients that are window-based (*xterm*, *xclock*, *xbiff*, etc.). It will not list the window manager or clients you want to run to set keyboard and display preferences, such as the pattern of the root window. (Chapter 14, *Setup Clients* explains how to set such preferences using the *xset*, *xsetroot*, and *xmodmap* clients.)

By default, *xlsclients* lists the clients running on the current screen of the display corresponding to the DISPLAY environment variable (almost always the local display). If your display is composed of multiple screens, you can list the clients running on all of them by using the `-a` command-line option. Note, however, that the output will not distinguish between the screens (though the command lines may contain a `-display` option revealing this information). If you have multiple screens, you're probably better off running *xlsclients* on each screen sequentially.

You can list the clients running on another physical display by using the `-display` option. See Chapter 3, *Working in the X Environment*, for more information about `-display`.

The bottom line is that you can use *xlsclient*'s output as the *foundation* for your session file, but you will need to add to it. See Appendix A for more detailed instructions on setting up a user session.

With the `-l` option (indicating long), *xlsclients* generates a more detailed listing. Example 8-4 shows the long version of the listing on the previous page.

Example 8-4. Long xlsclients listing

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

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

As we'll see in Chapter 10, many clients, including *xterm*, allow you to specify an alternate name for a client and a title for the client's window. If you've specified a title, it will appear in the *xlsclients* `Name` field. If you haven't specified a title but have specified a name for the application, the name will appear in this field. Neither of the clients in the sample display has been given an alternate name or title.

You use the instance and class resource names to specify default window characteristics, generally by placing them in a file in your home directory. This is described in detail in Chapter 11, *Setting Resources*.

Generating Information About the Display: xdpyinfo

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

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

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

See Volume One, *Xlib Programming Manual*, for insights into some of the other information provided by *xdpyinfo*.

User-contributed Clients

In addition to the clients in the standard MIT X distribution, there are many user-contributed clients available in the X source tree, distributed over Usenet and perhaps included with various commercial distributions. If you have access to Usenet, the newsgroup *comp.windows.x* contains voluminous discussions of X programming and the newsgroup *comp.sources.x* contains sources.

For example, several window managers are available. Some of the more popular window managers are:

twm	The tab window manager, which is the standard window manager shipped with MIT's X distribution.
awm	The Ardent window manager (written by Jordan Hubbard of Ardent Computer Corporation).
rtl	The tiled window manager (written by Ellis Cohen at Siemens Research & Technology Laboratories, RTL).
olwm	OPEN LOOK window manager (developed by AT&T).

A version of the OPEN LOOK window manager is available as a user-contributed client.

Commercial products (such as spreadsheets, word processors, and graphics or publishing applications) based on the X Window System are also becoming available.

In the next sections, we describe how to use three public domain programs that can assist you in using color on your display. *xcol* allows you to see how X's standard colors will look on your monitor. *xcoloredit* and *xtici* allow you to create shades of your own, which you can then use for backgrounds, borders, etc. See Chapter 10 for a discussion of the color options accepted by most clients and Chapter 12, *Specifying Color*, for an overview of X's color capabilities and the standard colors provided.

Previewing Colors for Your Monitor: xcol

Among the more useful user-contributed clients is *xcol*, developed by Helmut Hoenig.* The standard colors the X server recognizes are listed in a file called *rgb.txt*, which is generally

*Although this application has not been rewritten for Release 5, we've found that the latest version of the client does work with an R5 server.

stored in */usr/lib/X11*. These standard colors are not portable. A color can look different on different monitors, perhaps much different than the color name suggests. Hypothetically, you could spend a lot of time testing the various colors provided. The *xcol* client allows you to see how the standard colors look on your display before you specify the colors for a client. Once you've selected certain colors, *xcol* can also assist you in editing the color specifications in your *.Xresources* file.

To run the *xcol* program, simply enter:

```
% xcol &
```

A window titled ColorView will be placed on your screen. The ColorView window displays the outline of a cube containing scattered pixels of the available colors, almost like a universe of colored stars. The position of each of the colored pixels in the cube represents its RGB value (see Chapter 12, *Specifying Color*).

In many cases, a primary shade is associated with several subshades, which are distinguished from the primary shade by a number appended to its name. For example, you can specify the color dark sea green, and also DarkSeaGreen1 through DarkSeaGreen4. Within *xcol*'s ColorView window, colors with the same name but different RGB values (signaling different intensities) are represented by a single pixel.

The pixels are not labeled but you should be able to distinguish basic colors on a good quality color monitor. If you place the pointer on any of the pixels, a small box containing the color name will be displayed. The color name appears in white and the border of the box appears in the color specified. If the pixel represents several associated colors of differing intensities, the box will also contain a spectrum of those colors (though the individual shades are lumped under the primary name). By moving the pointer onto various pixels, you should be able to get an idea of how certain colors look on your display.

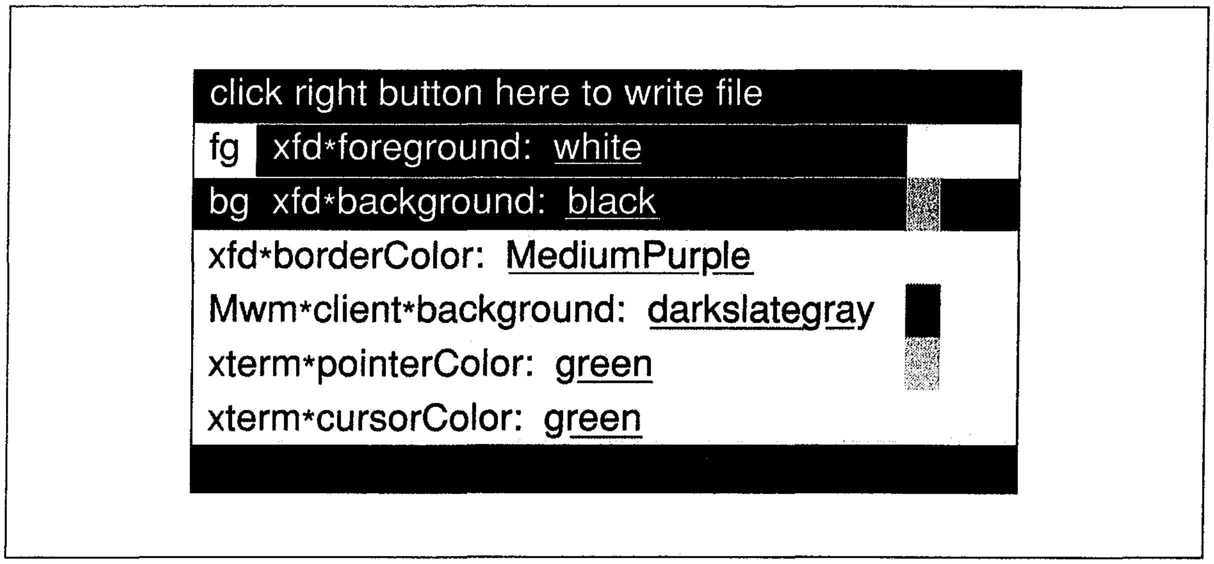
Some areas of the window are more cluttered with pixels than others. In these areas, you may not be able to distinguish individual pixels. However, if you move the pointer slowly over these "bunches," the individual color names will be displayed, outlined in the color.

While the pointer focus is directed to the ColorView window, you may notice the rest of the display becomes slightly darkened. This darkening happens because *xcol* provides its own colormap, different from the default. It is a normal effect and will stop when you switch the focus to another window.

In addition to letting you preview colors, *xcol* can also be used to edit color resource specifications. If you want to edit the color specifications in your *.Xresources* file, start the client using the command line:

```
% xcol ~/.Xresources &
```

This time two windows will be displayed: the ColorView window and a second window titled TextView, which contains the specifications pertaining to color from the *.Xresources* file, as in Figure 8-16.



```
click right button here to write file
fg  xfd*foreground: white
bg  xfd*background: black
xfd*borderColor: MediumPurple
Mwm*client*background: darkslategray
xterm*pointerColor: green
xterm*cursorColor: green
```

Figure 8-16. xcol's TextView window

The size of the TextView window depends on the number of color specifications in the *.Xresources* file. Though you can't tell from our black and white illustration, each specification in the TextView window appears in the color it names (the color is the foreground color of the text line). If you click the second pointer button on a specification line, a reverse video effect takes place: the named color becomes the background and the previous background color (gray by default) becomes the foreground (text) color. An "R" appears to the left of the text line, indicating that reverse video is enabled. Although reverse video is not necessary, it sometimes provides a better look at a color than the default display—and also a better look than the boxes surrounding color names in the ColorView window. Reverse video display is a toggle: if you want to return to the default display, click the second pointer button on the text line again.

By using the pointer on both the TextView and ColorView windows, you can change the colors specified in the *.Xresources* file. To select the resource to change, place the pointer on the corresponding line in the TextView window and click the first button. The selected line will be outlined in the current foreground color.

Once you've selected a resource, you can change the corresponding color value:

1. Place the pointer on a pixel in the ColorView window; wait until the color name box is displayed.
2. While the pointer rests on the pixel, click the first button.

The color value of the resource in the TextView window is changed to the color you select. The named color is also displayed as either the foreground or background color of the text line—background if reverse video was active. You can change the color any number of times without saving changes to the text file—just to take a look at some colors.

Now let's consider a practical example. Say we want to change the following specification in our sample resource file:

```
Mwm*client*background: darkslategray
```

which makes client window frames dark slate gray. First we select the resource by clicking on it with the first pointer button. Then we move the pointer to the ColorView window and search for an alternative color that would be good for the *mwm* frame. Moving the pointer among the colored pixels in the ColorView window, we settle on medium orchid. While the pointer rests on the medium orchid pixel and the box enclosing the color name is displayed, we click the first pointer button. The resource in the TextView window changes to reflect our choice:

```
Mwm*client*background: mediumorchid
```

and the color in which the line is displayed also changes from dark slate gray to medium orchid.

If the file displayed in the TextView window includes background and foreground specifications for the same resource, those resources are grouped together, and the letters "fg" and "bg" appear to the left of the foreground and background resources, respectively. These associated resources are displayed using the foreground and background color specifications they name. In our sample TextView window, the following resources are grouped:

```
xfd*foreground: white
xfd*background: black
```

These resources set the foreground and background colors for the *xfd* font displayed, described in Chapter 6, to white and black respectively. Thus, in the TextView window these specification lines appear in white with a black background.

To switch the colors specified by a foreground/background pair, place the pointer on either resource line and click the second button. Our sample resources would be changed to:

```
xfd*foreground: black
xfd*background: white
```

and the resources in the TextView window would also switch to black on white.

Keep in mind that you can change only one of the associated resources if you want, by using the method described previously.

Once you've selected colors you like, you can save the changes to the text file by placing the pointer in the horizontal bar at the top of the TextView window right below the frame and clicking the third button. The bar contains the message:

```
click right button here to write file.
```

When you click the third pointer button on this bar, *xcol* beeps and asks you to confirm the choice by displaying:

```
confirm writing with right button!
```

To write the file, click the third button on the bar again. The *.Xresources* file is saved and the following message is displayed in the horizontal bar:

```
    file written with backup.
```

The previous version of the *.Xresources* file is saved as a backup and given the filename *.Xresources~*. To restore the old settings, simply rename the backup file *.Xresources*.

If you want to cancel writing the file, click any button other than the third on the horizontal bar and you will get the message:

```
    writing aborted.
```

To quit the application, focus input on either the TextView or ColorView window and type q. Be aware that *xcol* will allow you to quit without saving changes and will not inform you.

Creating Colors of Your Own: xcoloredit and xtici

Several public-domain clients are available to help you create your own colors. In the following sections, we'll consider two of them: *xcoloredit* and *xtici* (also known as the TekColor™ Editor).

Of the two, *xcoloredit* is simpler to use. It works on the principle that you can "mix" red, green, and blue to create a color. (See Chapter 12, *Specifying Color*, for an explanation of the RGB color model.) The primary limitation: *xcoloredit* outputs only RGB color values, which are not portable. (They look different on different monitors.)

xtici is more complicated; to use it effectively, it's helpful to understand some of the more "scientific" terms used to describe color, namely *hue*, *value*, and *chroma*. The average person uses "blue," "light," "dark," "bright," etc. However, you can also describe (and specify) a color in terms of the following characteristics:

Hue	Generally speaking, the shade (e.g., red).
Value	A range of the hue from light to dark. The lightest possible shade of any hue is white; the darkest is black.
Chroma	Also called *saturation*. The amount of the hue present (roughly speaking, the hue's intensity).

As we'll see, when using *xtici*, you create a color by manipulating these three factors. However, these characteristics don't just apply to colors created using *xtici*—they can be used to describe any color. Though you'll undoubtedly think in terms of red, green, and blue when mixing colors with *xcoloredit*, the program does allow you to adjust the color based on hue, value, and chroma as well.

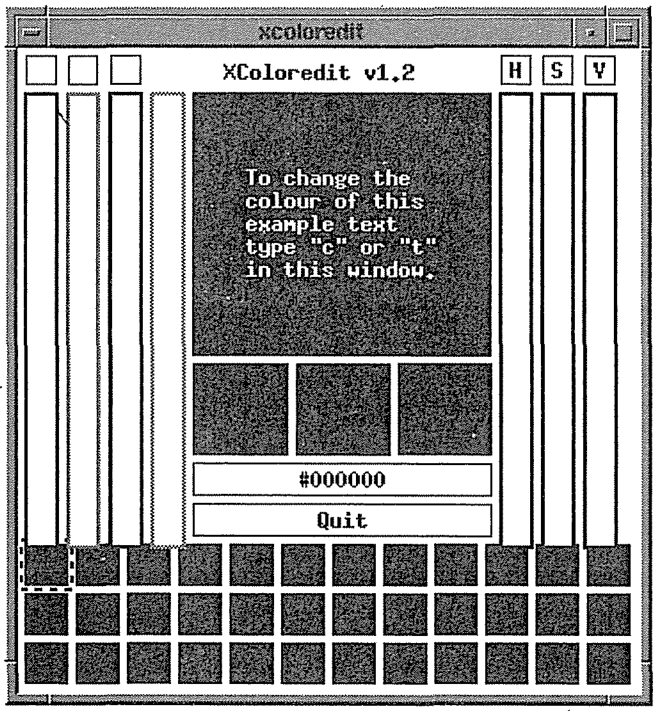
But before you can really understand what all this means, you'll need to run the color editors and play with them a bit. If you want more background information before this hands-on session, see Chapter 12, *Specifying Color*.

xcoloredit

The *xcoloredit* program is a public-domain client that can assist you in "mixing" your own RGB color specifications.* You can copy and paste a value *xcoloredit* supplies to specify a color on the command line or in a resource file. You might also pair this value with a name and place it in the RGB or Xcms color database, as described in Chapter 12, *Specifying Color*. Then you can specify the color by its name on the command line or in a resource file. But for now, let's just see how to mix some colors. *xcoloredit* is a simple and highly useful program with a fairly intuitive interface. You should quickly be able to use it effectively. To start the program, simply enter:

```
    % xcoloredit &
```

Figure 8-17 shows the initial state of the *xcoloredit* window. The various parts of the application are labeled.



To change the
colour of this
example text
type "c" or "t"
in this window.

Figure 8-17. Initial xcoloredit window

Don't be deceived by this blank looking window. *xcoloredit* lets you mix colors as easily as a kid mixes finger paints—but a little more neatly.

xcoloredit is available via anonymous *ftp* from *export.lcs.mit.edu* as */contrib/xcoloredit.tar.Z*. See Appendix H, *Obtaining Example Programs*, for instructions on retrieving files using *ftp*. For instructions on compiling public-domain programs, see Volume Eight, *X Window System Administrator's Guide*.

The idea behind *xcoloredit* is that you can mix parts of red, green, and blue to make a color that is displayed in the large central square—we'll call this the mixing area. Initially, the mixing area is black and displays the following text in white:

```
To change the
colour of this
example text
type "c" or "t"
in this window.
```

Below the mixing area are three smaller squares, which will show you how much red, green, and blue you have added. The red square is on the left, green in the middle, and blue on the right. To begin with, these three squares are also black.

Below the mixing area and the three component squares are 36 much smaller squares called "color cells." The color cells allow you to save up to 36 colors (and their hex values). More about this later. When you begin, all of the cells are black and the first cell (the upper-left one) is surrounded by a red dotted line.

To mix a new color in the large square (mixing area), you can use the three vertical bars on the left side of the window, which are outlined in red, green, and blue, respectively. These color sliders or scrollbars operate exactly like the Athena scrollbar (explained in Chapter 5, *The xterm Terminal Emulator*). Though you cannot tell immediately, each scrollbar has a "thumb" which can be dragged up from the base of the bar to add more of the color it represents. (Initially, no colors are present, so the mixing area is black.)

To get an idea how this works, place the pointer at the base of the red scrollbar (the leftmost one). When the thumb is accessible, the pointer changes to a vertical double-sided arrow. Press and hold down the second pointer button. The pointer again changes, this time to an arrow pointing right. Now drag the (pinkish) thumb toward the top of the scrollbar. As you drag up, you're in effect adding more and more red. As the thumb moves, notice that five other parts of the display are being constantly updated:

1. The mixing area is changing to reflect the amount of red you're selecting.
2. The red square (the leftmost square beneath the mixing area) also displays the amount of red you're selecting.
3. The hexadecimal window beneath the three color squares is being updated to show the numeric color value.
4. The dotted line surrounding the first color cell becomes a solid line. The color cell displays what is in the mixing area.
5. On the right side of the window, three additional sliders (labeled H, S, and V) move concurrent with the red slider.

To mix a color, you'll probably want to play with the green and blue sliders as well. Note that if you also select green and/or blue, the mixing area and the first color cell display the new combination, while the smaller squares below the mixing area display the component colors—to show you the amount of each color you're adding.

If you click the first pointer button in the hexadecimal window, the text becomes the PRIMARY selection (the hex window becomes reverse video to signal this); you can then paste the hex value into an *xterm* window by clicking the second pointer button. This is handy for transferring the text to the command line or to a resource file.

For example, in Figure 8-18 we've mixed a color like pea soup; it has the hex value #a8bc49.* (Since these illustrations are black and white, I'm afraid this will take some imagination!)

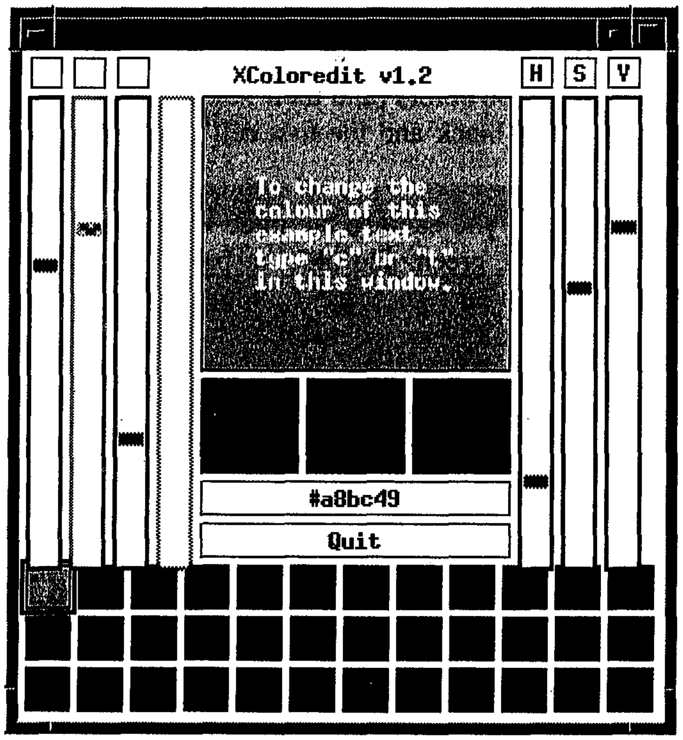

XColoredit v1.2
H
S
V
To change the colour of this example text, type "c" or "t" in this window.
#a8bc49
Quit

Figure 8-18. Mixing pea soup with xcoloredit

In Figure 8-19, we click the first pointer button in the hex window, which becomes reverse video as the value is made the PRIMARY text selection.

Then we can paste the hex value on the command line or in a resource file. For instance, we can type:

```
    % xload -fg
```

*This RGB color value may produce a different shade depending on the display hardware and the X server. See Chapter 12, *Specifying Color*, for more information.

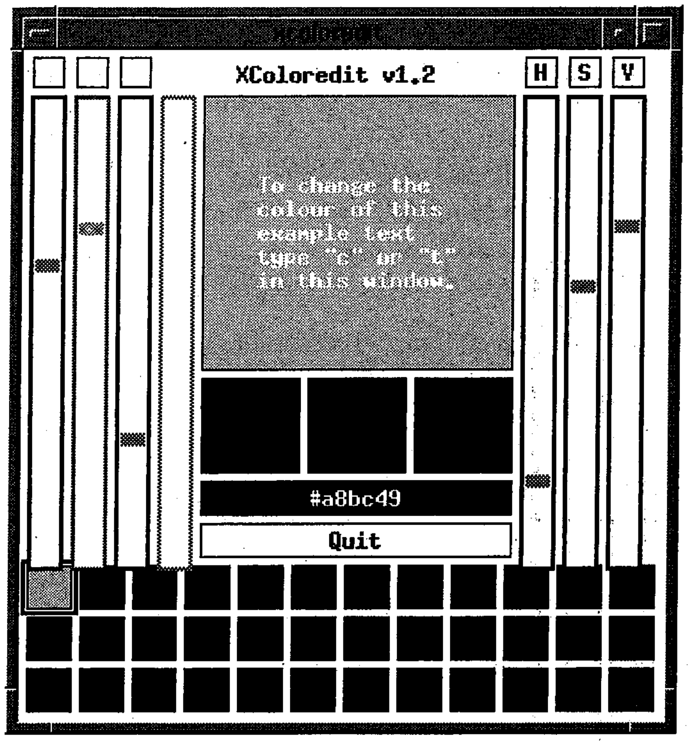

XColoredit v1.2
H
S
V
To change the colour of this example text type "c" or "t" in this window.
#a8bc49
Quit

Figure 8-19. Copying the hex value into memory for pasting

Then click the second pointer button to paste the hex value for our new pea soup color (and add an ampersand to run the process in the background):

```
% xbiff -fg #a8bc49 &
```

This command line should create an *xbiff* window with a pea soup colored mailbox.

We could also paste the hex value into a resource file like *.Xresources*. The following line specifies that all *xbiff* windows use the pea soup color as the foreground color.

```
XBiff*Foreground:  #a8bc49
```

See Chapter 11, *Setting Resources*, for more information.

Saving Multiple Colors Using the Color Cells

Once you've mixed one color, you can retain the image of that color in the first color cell and mix a second color which appears in the second cell. The hex values will also be saved so you can copy and paste them.

To begin, place the pointer on the second color cell (the one immediately to the right of the first) and click the first pointer button. The second cell becomes surrounded by a dotted outline; it also changes from displaying black to displaying the color in the first cell (and the mixing area).

Now you can begin mixing a second color, just as you did the first, by dragging the color sliders. As you move the sliders, the five parts of the window described previously are updated. Most significantly, the mixing area and the second color cell will display the new color, and the hex window will display its numeric value.

For our second color we've come up with a very bright blue (remember: imagination) with the hex value #09e5fb, as in Figure 8-20.

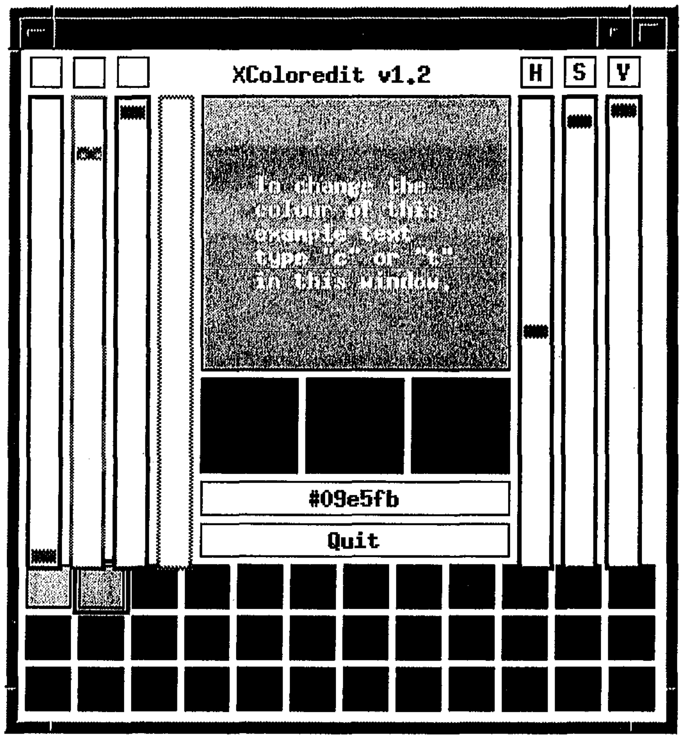

XColoredit v1.2
H
S
V
To change the colour of this example text type "c" or "C" in this window.
#09e5fb
Quit

Figure 8-20. Mixing a second color

You can go back to the first color (display it in the mixing area) by placing the pointer on the first color cell and clicking the first button. All parts of the window go back to the state associated with the first color: the mixing area displays that color, the first color cell is outlined, the hex window displays its value (so you can copy and paste it), the sliders indicate the red, green, and blue components, etc.

To switch back to the second color, click on the second color cell. The display now reflects the second color.

If you want to use all of the color cells, you can actually mix and save 36 colors! Then you can "browse" through them by clicking on associated color cell. In effect, *xcoloredit* serves as a sort of color clipboard.

Moving on to the next color cell doesn't mean the previous one is frozen as it is. You can select any color cell and continue to edit it at any time. Or you can consider a cell to be static and move on to edit within another cell.

What Are the H, S, and V Sliders For?

Anyone who's mixed finger paints can get used to the red, green, and blue sliders fairly easily. Most people can conceive of color as this sort of mixture. When you move any one of the red, green, and blue sliders, on the right side of the window, three additional sliders (labeled H, S, and V) move concurrently. While the red, green, and blue sliders (obviously) represent the amount of each of those colors, the H, S, and V sliders represent other characteristics used to "measure" color: hue, saturation, and value (also sometimes called chroma).

These characteristics are described in the introduction to this section. If you want to, you can mix a color in the *xcoloredit* window using the H, S, and V sliders (the red, green, and blue sliders will move concurrently), but the program is not actually intended to be used in this way. Hue, saturation (or chroma), and value probably won't be very meaningful (or relevant)—unless you experiment with the TekColor editor, *xtici*.

Quitting xcoloredit

To quit the application, click the first pointer button on the Quit button, which is located just below the hex window.

xtici (The TekColor Editor)

For many users, *xtici* will be somewhat unintuitive, but it offers a significant advantage over *xcoloredit*—namely, that it can provide color values in two portable formats (as well as the non-portable RGB format). Playing with *xtici* for a while should help you relate the characteristics hue, chroma, and value to the typical terms with which people describe color: "red," "green," "light," "bright," etc.

Note that *xtici* is useful because of the X Consortium's adoption (in Release 5) of the X Color Management System (Xcms), developed by Tektronix. Xcms is intended to overcome the limitations of the (still available) RGB model by providing *device-independent* color. A device-independent color looks the same on any monitor.

Xcms accepts color values in several different formats, called *color spaces.* Most of these color spaces describe color in a device-independent manner, using scientific terms and values commonly applied to color (like hue, value, and chroma). Xcms also recognizes the non-portable RGB color format. (For more information, see "The Xcms Color Spaces" in Chapter 12.) *xtici* interprets and also outputs color values in two of the Xcms portable color spaces and in RGB format.

xtici has an elaborate interface, complete with several menus. We'll just take a look at a few of the features you'll need to create your own colors. For more information, see the *TekColor Editor Reference Manual*, distributed in PostScript form with *xtici*.

To run the program, simply enter:

```
% xtici &
```

Figure 8-21 shows a typical *xtici* window with some of the more significant features labeled.

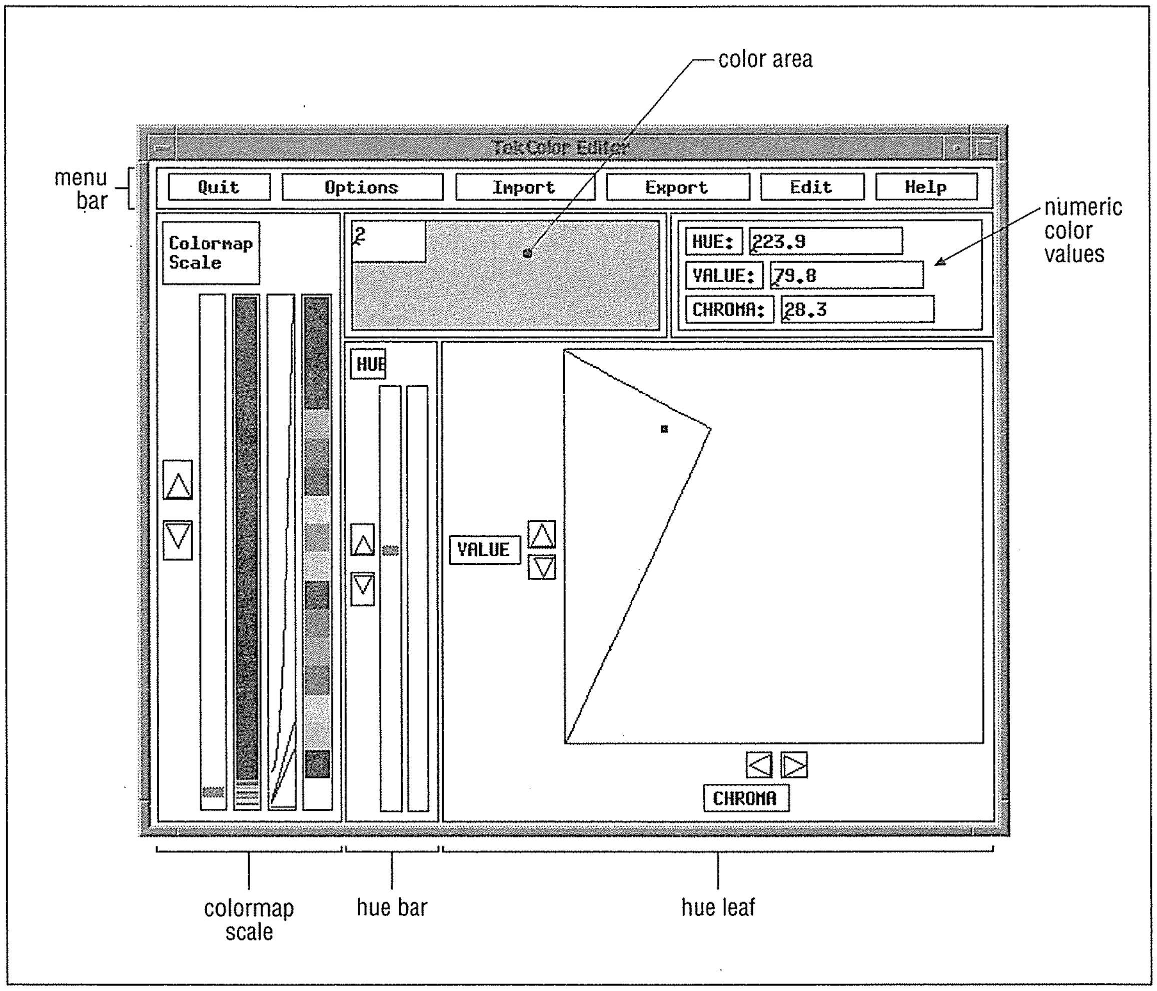


Figure 8-21. Initial xtici window

For our purposes, the most important parts of the window are:

- The Quit, Options, and Edit menus on the menu bar.
- The color area, in which the current shade is displayed.
- The numeric color values, which can be used to specify the current shade. (The Edit menu allows you to select these values to paste on the command line, in a resource file, in an Xcms color database, etc.)
- The Colormap Scale, from which you can change the hue to be edited.
- The Hue Bar provides a sampling of a wider range of hues than is initially visible in Colormap Scale. The Hue Bar also allows you to change the hue to be edited.
- The Hue Leaf, which shows the value and chroma range for the selected hue. You can adjust the value (lightness) and chroma (intensity) from this area.

The color area shows an initial color, the numeric values for which appear in the boxes immediately to the right. *xtici* allows you to change or edit this color in a variety of ways. Some of the editing methods involve dragging or clicking on graphical elements in the window, but you can also change the color in the color area by entering other numeric values in the boxes to the right. If you edit using the former style, the numeric values will change concurrently.

Because *xtici* provides many different ways to change the color in the color area, editing can be complicated. We'll show you some of the basic editing strategies in the next few sections, but regardless of the method you choose, you'll probably want to perform the following tasks:

1. Select a hue.
2. Adjust the value and chroma of the selected hue, if desired.
3. Select the numeric value of the color (using the Edit menu) in order to specify it on the command line, etc.

Choosing a Hue to Edit

When you first run *xtici*, the color in the color area will probably be a color that is currently being used elsewhere on the display. You can work from this initial hue—adjust its value (or lightness) and chroma (or intensity)—or select another hue and edit that.

In Figure 8-20 our initial hue is the standard RGB color named sky blue. (See Chapter 12 for a discussion of the RGB color model and the RGB color database.) The box to the right of the color area contains numeric values that correspond to this color. By default, the color is expressed as three numbers corresponding to hue, value, and chroma. This so-called TekHVC format (or color space) is portable. *xtici* also recognizes and can return values in another portable format, introduced by the prefix CIEuvY, as well as in the non-portable RGB format.

Notice that the hue in the color area also appears in the so-called Colormap Scale area of the *xtici* window. Within the Colormap Scale area are four parallel vertical bars. The first bar is a slider. (You can use the slider or the up and down arrows to its left to adjust the hue in the color area.) The second bar shows a spectrum of some of the possible colors. (When you first run *xtici*, this second bar shows the colormap of the current screen—thus, the colors will be those the clients are using.) The fourth vertical bar shows a magnification of part of the second bar—specifically, the shade that also appears in the color area and the shades that surround it in the colormap. (The third bar simply contains lines indicating what part of the second bar is being magnified in the fourth.)

If you want to work from the hue in the color area (adjust its value and chroma), skip ahead to the section "Adjusting the Color with the Hue Leaf."

If you want to change the initial hue, there are several ways to do it:

- Click the first pointer button on the color you want from either the second or fourth vertical bars in the Colormap Scale area.