Using the X Window System

HP 9000 Series 300/800 Computers

HP Part Number 98594-90040

Hewlett-Packard Company
1000 N.E. Circle Blvd., Corvallis, OR 97330

Notice

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MANUAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Warranty

A copy of the specific warranty terms applicable to your Hewlett-Packard product and replacement parts can be obtained from your local Sales and Service Office.

The X Window System is a registered trademark of Massachusetts Institute of Technology.

UNIX is a registered trademark of AT&T in the USA and other countries.

© 1988 Hewlett-Packard

MANUAL COMMENT CARD

HP Part Number 98594-90040 E1288

Your comments and suggestions help us determine how well we meet your needs.

Using the X Window System

	Agree				Disagree
The manual is well organized.	○	○	○	○	○
It is easy to find information in the manual.	○	○	○	○	○
The manual explains features well.	○	○	○	○	○
The manual contains enough examples.	○	○	○	○	○
The examples are appropriate for my needs.	○	○	○	○	○
The manual covers enough topics.	○	○	○	○	○
Overall, the manual meets my expectations.	○	○	○	○	○

You have used this product:

__ Less than 1 week __ Less than 1 year __ More than 2 years

__ Less than 1 month __ 1 to 2 years

fold ——

Please write additional comments, particularly if you disagree with a statement above. Use additional pages if you wish. The more specific your comments, the more useful they are to us.

Comments: _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Please Tape Here

Please print or type your name and address.

Name: _____

Company: _____

Address: _____

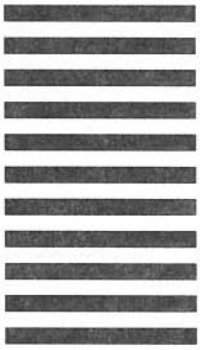
City, State, Zip: _____

Telephone: _____

Additional Comments: _____

Using the X Window System
HP Part Number 98594-90040
E1288

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 38 CORVALLIS, OR

POSTAGE WILL BE PAID BY ADDRESSEE

HEWLETT-PACKARD COMPANY
CWO PRODUCT MARKETING
1000 NE CIRCLE BLVD
CORVALLIS OR 97330-9988

Printing History

The manual printing date and part number indicate its current edition. The printing date will change when a new edition is printed. Minor changes may be made at reprint without changing the printing date. The manual part number will change when extensive changes are made.

Manual updates may be issued between editions to correct errors or document product changes. To ensure that you receive these updates or new editions, you should subscribe to the appropriate product support service. See your HP sales representative for details.

December 1988 ... Edition 2

Contents

1

How to Improve Your X Life

Welcome to graphical user interfaces ("windows") and to the X Window System version 11 (X11 or X) in particular. In this chapter you'll find out how this manual is organized and some of the conventions it uses. You'll also find some tips to make learning about X11 easier and to improve your X life thereafter.

How This Manual Is Organized

This manual is organized so that the less technical information comes first.

If you're new to computers, new to HP-UX, or have had some window experience – but never this much control of your screen environment – you'll want to read this chapter and chapters 2 and 3. You'll also find the glossary and the index helpful.

Chapter 1	Introduces this manual and gives some tips on HP-UX and networking.
Chapter 2	Explains the window environment and sets the stage for chapter 3.
Chapter 3	Provides a beginner-level introduction to using the X Window System.

If you're a system administrator or programmer – someone familiar with computers and how they operate – you'll probably be more interested in the more technical information in the second half of this manual.

Chapter 4	Explains how to run programs from the command line.

Chapter 5	Discusses customizing the X environment to suit your personal needs or the needs of the users who use your system.
Chapter 6	Offers a detailed explanation of the HP Window Manager.
Chapter 7	Provides information about customizing special X environments.
Chapter 8	Discusses printing and screen dumping.
Chapter 9	Discusses the use of Starbase graphics.
Reference	Contains "man" pages—the definitive description—for current X clients.

But please, System Administrator, don't just skim the man pages, tweak the `sys.Xdefaults` and `system.hpwmrc` files, and then bury this manual on a bookshelf. Your users, the people who depend upon you for support, could use this manual to make life a little easier for themselves—and for you. Make it available to them and encourage them to read it. As they become self-sufficient within their window environment, your support tasks become easier.

Ultimately, whether you're a new user or an experienced user, the purpose of this manual is to improve your X life.

Conventions

As you read this manual, notice the following typographical conventions:

Table 1-1. Typographical Conventions.

If you see ...	It means ...
`computer text`	This text is displayed by the computer. For example, `login:` is a login prompt displayed by the computer.
italic text	A book title, emphasized text, or text that you supply. For example, hpterm -fg *color* means that you type "hpterm -fg" followed by a color of your choice.
	You press the corresponding key on the keyboard. For example, CTRL Left Shift Reset means you hold down the CTRL key, the Left Shift key, and the Reset all at the same time.
`[ ]`	An optional parameter that can be left off if you don't need that functionality. For example, `xload [-rv] &` means that you must type "xload" but don't have to type "-rv".
`{ }`	A list containing *mutually exclusive* optional parameters. For example, `xset r { on / off }` means that option `r` can be set to either `on` or `off`, but not both.
bold text	The definition of this term follows. Additionally, the term is defined in the glossary.

Also, you can use the X Window System with either a two- or a three-button mouse by observing the following conventions:

Table 1-2. Mouse Buttons and Their Locations.

If you see ...	On a 3-button mouse press ...	On a 2-button mouse press ...
Select button	The left button.	The left button.
Alternate button	The middle button.	Both buttons simultaneously.
Menu button	The right button.	The right button.

Running HP-UX: Some Tips

If you are new to HP-UX and to the X Window System, take heart: You're not alone. A wide variety of users, many just like yourself, are currently learning HP-UX and X11. The next several paragraphs provide you with some information and some tools to facilitate the initial stages of learning.

What Is HP-UX and the X Window System?

HP-UX is Hewlett-Packard's implementation of the UNIX operating system. The operating system is the software that controls the operation of the computer system. HP-UX is a multi-user, multi-tasking environment. A multi-user environment means more than one user can be operating the system at the same time. A multi-tasking environment means that each of those users can run more than one program at a time.

The X Window System is a window environment. It turns your screen into a "root window" or "desktop" on which you can display smaller windows, each one the equivalent of a full-sized display terminal. Within the X11 environment you can run multiple tasks, viewing their progress in separate windows.

Why Background Processing Is Important

Your programs can run as either foreground or background processes. In any X11 terminal widow, you can only run one program at a time as a foreground process, but you can run as many programs as you like as background processes. To run a program as a background process, add an ampersand (`&`) to the end of the command line that starts the program. The ampersand tells the system that the program should be run in the background. This leaves the foreground free for you to issue more commands.

Take, for example, the following command:

```
xclock &
```

This command starts a clock. The `&` tells the system to display the clock, but as a background process, so you can use the foreground to enter more commands. Without the `&`, the clock would still display, but in the foreground. The window from which you issued the command would ignore everything else, including your keyboard commands, as long as the clock remained the foreground process. This could prove inconvenient, even to inveterate clock-watchers.

If you forget an `&`, you will need to stop that program to regain control of the foreground—a task not always easy to accomplish (see either "Exiting from the X Window System" in chapter 3 or "Stopping Programs" in chapter 4).

One last note on foreground and background: Don't confuse foreground and background *processing* with the fact that you can *color* the foreground and background of your windows. The foreground and background that you process are not the same foreground and background that you color. Foreground and background processes are activities of the computer; the foreground and background that you color are graphical elements that display on the screen.

Case Sensitivity and Other Typographical Tips

HP-UX distinguishes between uppercase and lowercase letters. A file named `.xdefaults` is *not* the same file as `.Xdefaults`. Use uppercase letters where indicated and *only* where indicated.

Also, the number "1" (one) looks an awful lot like a lowercase "l" (el) to our human eyes. The system, however, can readily distinguish the difference and often seems to do so with a vengeance.

Don't confuse the "0" (zero) with the upper case "O" (oh) for the same reason.

White space (extra spaces or tabs) at the end of a command line in a text file sometimes alters the meaning of the command. Files such as `.rhosts` are especially vulnerable. After modifying a file, check for unwanted white space.

And finally, *watch your spelling.*

Working with HP-UX Manuals

HP-UX manuals typically have a section devoted to reference information. This section contains "man" (manual) pages that provide specific information about a command, function, or program. The man page is the most definitive source of information. You will find man pages in the reference section at the back of this manual.

Logging In to HP-UX

Most HP-UX systems require you to log into a system before you gain access to the resources available on that system. The administrator of the system must provide you with a login account. When you have a login, you will be able to log into that system by providing your login name and your personal password. When you are logged in, you may use the resources available such as the X Window System.

Note that on some systems the system administrator may have configured your login process so that you automatically start your X environment.

For More Information

Several beginner's guides come with your computer system.

Table 1-3.

To learn about ...	Look through this guide ...	HP Part Number
Using the HP-UX operating system concepts and commands.	*A Beginner's Guide to HP-UX*	98594-90000
Using shells to increase performance.	*A Beginner's Guide to Using Shells*	98594-90020
Editing commands for the `vi` editor.	*A Beginner's Guide to Text Editing*	98594-90010
Customizing your own X Window System environment.	*A Beginner's Guide to the X Window System*	98594-90001

If you are new to the system, taking the time to study these guides will help clarify questions you may have.

There is also a great deal of information available about the HP-UX operating system in the *HP-UX Reference* volumes that accompany the operating system.

Additionally, information about programming in the X Window System environment is available in the following manuals:

Table 1-4.

To learn about ...	Look through this manual ...	HP Part Number
Writing and using widgets in application programs.	*Programming with the HP X Widgets and the Xt Intrinsics*	98794-90000
Fortran Bindings and Native Language I/O systems.	*Programming with the Xrlib User Interface Toolbox*	5959-6160
Writing graphics programs for X.	*Programming with Xlib*	98794-90010

Finally, depending on your needs, the following books about the X Window System might prove useful:

- *Introduction to the X Window System* by Oliver Jones. Prentice Hall, Englewood Cliffs, NJ:1989.
- *Xlib Programming Manual for Version 11* by Adrian Nye. O'Reilly and Associates, Newton, MA:1988.

- *Xlib Reference Manual for Version 11* edited by Adrian Nye. O'Reilly and Associates, Newton, MA:1988.
- *X Window System User's Guide* by Tim O'Reilly, Valerie Quercia, and Linda Lamb. O'Reilly and Associates, Newton, MA:1988.

Where to Go Next

Now that you've finished these preliminaries, you have a choice. If you feel comfortable with (or aren't interested in) an explanation of graphical user interfaces, skip chapter 2 and read chapter 3 on how to use the X Window System.

If you've had some experience with graphical user interfaces and the X Window System in particular, you might want to skip all the way to chapters 4 and 5 to find out how to run X11 clients and customize your X11 environment to your individual needs.

2

Understanding Window Systems

This chapter is written for new users. If you're not familiar with HP-UX or window environments, this chapter's for you. It describes the following key elements:

- Basic window concepts.
- A typical X Window System environment.
- An example of a distributed computing environment.

This chapter demonstrates the power and the flexibility of the X Window System.

What Is the X Window System?

The X Window System is a **graphical user interface,** a way of communicating with your computer using visual images (graphics). You can better understand the importance of the X Window System and why its possibilities are so exciting if you compare it to the "traditional" user interface, the command-line prompt.

In contrast to the austerity of the command-line prompt, the X Window System offers a visually rich connection to your computer. This connection, the user interface, is characterized by easily recognizable graphical features: windows, selection menus, and icons.

X11 surrounds your interaction with the computer system in a visual metaphor more intuitively meaningful—especially to novice users—than the command-line prompt with its often esoteric commands and obscure parameters. X11 provides you with a friendly, easy-to-use work environment.

X11 Is Based on the Server-Client Interaction Model

The X Window System is based on a **server-client interaction model.**

The **server** is really what you "start" when you "start X11." The server controls all access to input devices (typically a mouse and keyboard) and all access to output devices (typically a display screen). You can visualize its position in the scheme of things by thinking of it as standing between the programs you run on your system and your system's input and display devices.

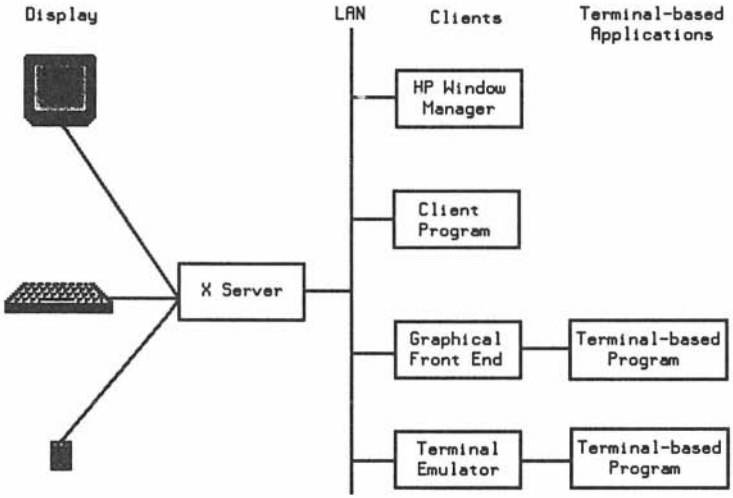


Figure 2-1. The Server Controls Display Access.

A **client** is any program written especially to run with the server. Another way of looking at it is to view the client as "window smart." Clients know about windows and how to make use of them. All other programs are **non-clients,** programs that don't know how to make use of windows.

Multi-Tasking Makes X11 a Powerful Tool

Part of the X Window System's power comes from the computer system's **multi-tasking** ability. Multi-tasking is the ability to execute several programs simultaneously. Each program is a separate task (process). The X Window System brings multi-tasking out of the realm of the power user and into the hands of the novice user in search of increased efficiency. In your X11 environment, you run each program in a separate window as a separate process. Windows may overlap on the screen, but their processes don't interfere with each other.

For example, you could have the system recalculate a large spreadsheet in one window while you shift your attention between editing a monthly report in a second window and answering your electronic mail in a third. Each program normally has a main window for visual interaction, and each window has its own input and output.

You focus your attention on a particular window by moving the mouse pointer into that window and pressing the select button. The window thus pointed to becomes the **active window.** While you focus on one window, other windows continue running unattended or wait for your input.

Multi-tasking is possible in part because of the way the computer system divides all processes into **foreground processes** and **background processes.** Background processes are the ones that run unattended or wait until they get your input. You can have as many background processes running in a window as you like. A foreground process is the process that has the window's attention at the moment. You can have only one foreground process running in each window.

The **ampersand (&)** at the end of a command line controls foreground and background processing.

X Allows Both Local and Remote Access

Any computing environment allows you **local access,** the ability to run a program on the computer in front of which you're sitting. Networked computing environments also allow you **remote access** to programs, the ability to run a program on a computer *other* than the one at which you're sitting.

Using the X Window System, you can run programs both locally and remotely at the same time. You also have greater control over where the output displays. If you wish, you can run a program locally and display the output on the screen of a remote system; or the opposite, run a program remotely and display the output in a window on your screen; or run a program remotely and have it display on yet another remote screen.

The X Window System Allows Multi-Vendor Networking

A final feature of X11 worth mentioning is the X Window System's acceptance as an industry standard for UNIX operating system network protocol. Since all X Window System hardware and software vendors communicate using the X protocol, any program from any vendor can be run remotely and viewed on your local system. Thus, computer networks composed of hardware and software from multiple vendors, instead of being a "nightmare of incompatibility," become powerful resources for specialized applications, allowing the user to select the best hardware and software for the application without compromising performance for compatibility.

The Parts of a Typical X Window System

Your personal window environment can be relatively simple or rather elaborate. The details depend on your personal computing needs, the programs you use, and how you customize three X Window System configuration files. However, all X Window System environments have the following features in common:

- Computer hardware (a system) on which to run the software.
- An X server program to control communication between the display and client programs.
- A window manager to control the display's window environment.
- Application programs to provide useful services.

The Computer Hardware System

The hardware system consists of several components:

- System Processing Unit (SPU).
- Hard disk.
- Keyboard.
- Mouse.
- Display screen.
- Connection to a Local Area Network (LAN).

The SPU Does the Computing

The System Processing Unit or SPU is the "brains" of the computer. The SPU contains the logic circuitry, which is driven by the software and performs all the processing that takes place. The SPU of your system runs the X server that provides your window environment, takes care of foreground and background processing, and controls local and remote accessing of your system's resources. Using X, you can run programs that are stored on your own hard disk (local processing) or that are stored on someone else's hard disk using their SPU (remote processing).

The Hard Disk Stores Data

The hard disk stores programs and data files. No processing takes place on the hard disk, only storage. Some HP 9000 Series 300 configurations are called **diskless clusters** because groups of users share the same hard disk.

The Keyboard Enters Text

The keyboard is an **input device,** a device used to put information into the computer. This information could be the text of a letter or the next command that the computer should execute, depending on whether you type the text into a file or on the command line.

Although the keyboard is frequently used in conjunction with a mouse, it does not need to be. You can configure your X11 environment so that you can use the keyboard for both text entry (its usual purpose) and for pointing and

selecting (the mouse's usual purpose). For example, this **mouseless operation** would be beneficial in any situation where desk space was at a premium.

The Mouse Points and Selects

The keyboard enters characters; the mouse points and selects. Sliding the mouse on your desktop moves the **pointer,** the current screen location of the mouse, on the screen. Using the mouse, you can point to an object on the screen, for instance a window, and select an action to perform, such as resizing. Selection is made by pressing the mouse's select button. As mentioned, however, mouse movements and button presses can be associated with keyboard key presses for mouseless operation.

The Screen Displays Output

The principal output device for the X Window System environment is the **display.** A typical display consists of one physical screen per mouse and keyboard. However, depending on the specialized nature of the application, a display may include as many as four physical screens, all using the same mouse and keyboard.

The **screen** is the physical CRT (Cathode Ray Tube) that displays what you type on the keyboard. The screen also shows you the position of the pointer and windows, and provides you with visible indications of the status of executing programs.

Conceptually, the screen becomes the **root window** when you start the X Window System. The root window contains all the windows, menus, and icons that compose the visual elements of your X11 environment.

Technically, the screen is known as a **bitmapped device** because the graphical elements (windows and icons) that it displays are stored by the computer as a **bitmap,** a pattern of bits (dots) that can be readily displayed as graphical images.

The LAN Connects to the Network

The LAN is composed of hardware and software. The hardware part connects your computer system physically (using a cable) to a network which includes other computer systems at your site and could encompass other networks

at different locations. The LAN enables you to take advantage of remote processing capabilities of X.

Other Pointing Devices

Although the mouse is the most common pointing device, the X Window System display server (the program that "runs" X on your system) supports other HP-HIL (Hewlett-Packard Human Interface Link) pointing devices, for example a digitizer tablet or track ball. References in this manual to mouse actions apply also to corresponding actions with other HP-HIL pointing devices.

The X Server Controls Communication

The server is the program that controls the screen, keyboard, and mouse, and processes all communication requests. The X server is really what runs when you "run X11." The server updates the windows on the screen as a client generates new information or as you enter information through an input device. All client programs communicate through the server.

Because the server controls communication with the display screen, it is sometimes called the **display server.** Either name is correct.

The Window Manager Controls Your Windows

The window manager is your main means of dynamically controlling the size, shape, state (icon or normal), and location of the windows on your screen. Several different window managers exist; the window manager for the Hewlett-Packard implementation of the X Window System is called (appropriately enough) the HP Window Manager (`hpwm`). The HP Window Manager includes:

- menus
- icons
- window frames

The System and Root Menus

One way that you can control the operation of your window environment is by choosing an action from a **menu.** A menu is a small window that contains a list

of selections—exactly like a restaurant menu. The HP Window Manager has two menus:

System menu	One for each window on your screen. A system menu controls the particular window to which it is attached.
Root window menu	The menu for the root window. The root menu controls actions that are generic and refer to no particular window.

The following figure shows a window with the system menu displayed and the "move" selection highlighted.

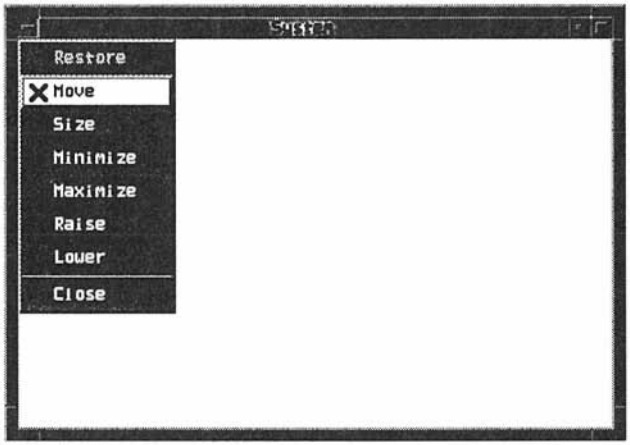


Figure 2-2. The System Menu with "Move" Selected.

You can configure your window manager to make life easier for yourself. For example, you can add a selection to the root menu that enables you to log onto a remote host and run an application automatically. You can also create submenus of related activities. One popular submenu is a list of remote hosts to log onto. Chapter 6 of this manual discusses configuring the HP Window Manager.

Icons

Because your display will often contain several windows, you may find it convenient to set aside a window you're not currently using *without stopping the processing in that window.* You do this by changing the window into an

icon, a small, easily identifiable graphic symbol that represents the window but takes little space on the screen.

The contents of an iconified window aren't visible. But you can quickly convert the icon to its original window representation whenever you wish to use the window again. Any processing that was occurring in the window as it was iconified continues as long as it doesn't require additional input from you. You won't be able to see output or enter input until you change the icon back into a window.

The figure below shows several icons, each representing a different type of client.

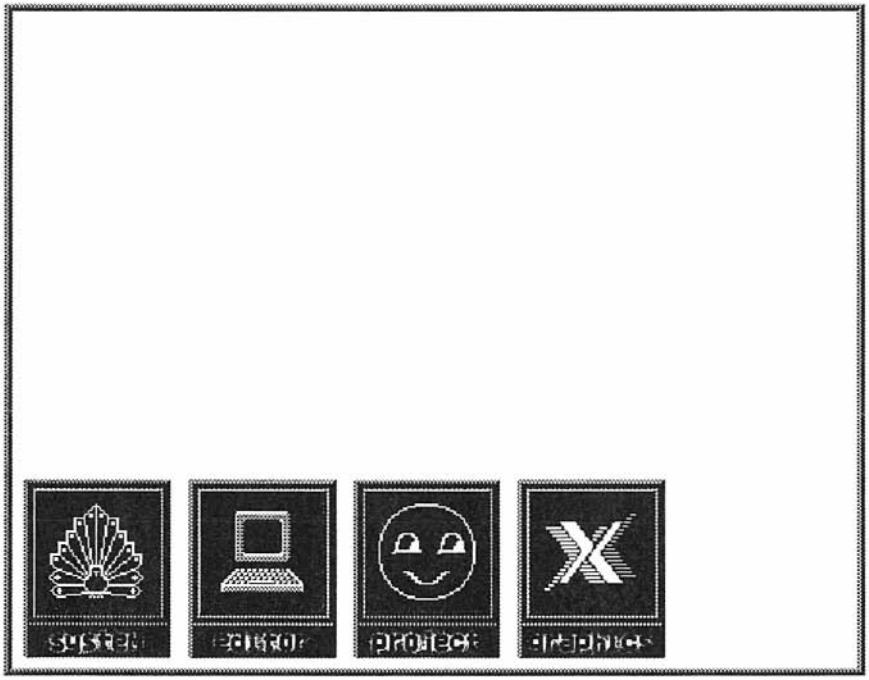


Figure 2-3. Icons Replace Windows Giving You More Room.

Window Frame Decoration

The HP Window Manager, unlike `uwm` (the other window manager included with the X Window System) provides a functional frame around each window

in the root window. The frame, sometimes called **window decoration,** consists of graphical control devices that enable you to display the window's system menu, maximize or iconify the window, or move and resize the window.

Application Programs Run in Your X Environment

An **application program** is a computer program that performs some useful function like word processing or data base management. The applications you run while you use X may be stored on the hard disk attached to your system or on the hard disk of a remote system. The X11 server communicates with application programs just as easily over the LAN as locally.

You can sort all application programs into two categories:

- Those that know about windows and incorporate windowing behavior into their own behavior (client programs).
- Those that don't know about windows and think that they must always be running on a separate terminal (non-client programs).

Window-Smart Programs Are Called Clients

A **client** is a program written especially for the X Window System. Clients are referred to as **window-based** programs. The window manager that controls the windows on your screen is a client. The windows themselves are clients. Clients are "smart" enough to create their own windows if they need to display output. Note, however, that not all clients create windows. Some clients (like `xwininfo` and `xmodmap`) are content to use an existing terminal emulation window in which to display their output.

Terminal-Based Programs Must Be Fooled

Non-client programs know nothing about windows. They are designed to run alone on display screens or "terminals" and are, therefore, referred to as **terminal-based** programs. Terminal-based programs must have windows created for them so that they can run in a window environment. They are thus "fooled" into operating in the window environment.

You can operate terminal-based programs in the X Window System by using a client program called a **terminal emulator** to provide a window. You start the non-client program in that window. The terminal emulator "fools" the

non-client into thinking that it is running on a "real" terminal instead of a window imitating a terminal. This has led some people to describe non-client programs as "window dumb."

The X Window System provides two terminal-emulator clients: **hpterm** and **xterm**. When either is run, it creates a window to emulate a display terminal. A terminal-based program runs happily in this window, acting exactly as if running on a terminal.

The following diagram shows the components of a system running X.

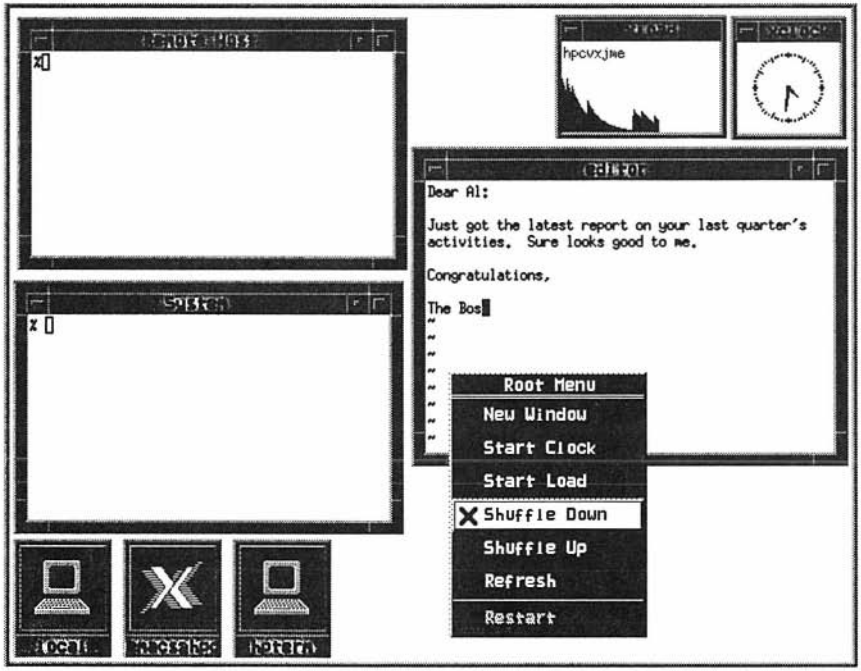


Figure 2-4. Typical Components of an X Window System.

The Distributed Computing Environment

A Distributed Computing Environment (DCE) is a group of computer systems joined together into a network. Resources resident on one system are available to all systems. As mentioned earlier, a system that uses X11 is usually connected to a LAN. The LAN provides the link to programs that are resident on physically separate (remote) systems.

X11 really doesn't care where a program is—it simply communicates to the program via the LAN connection. This structure permits you to operate S300 and S800 systems at a strictly local level with all client programs residing locally, or at a networked level with some programs running at the local level while others run on remote systems.

In addition, another system on the LAN can run programs that reside on your S300 or S800 and direct the visual output to *any* screen on the network.

A distributed computing environment, in other words, enables the best possible allocation of processing resources within the existing hardware environment.

The figure below shows a distributed computing environment that provides a number of resources to users who are connected to the LAN and running X.

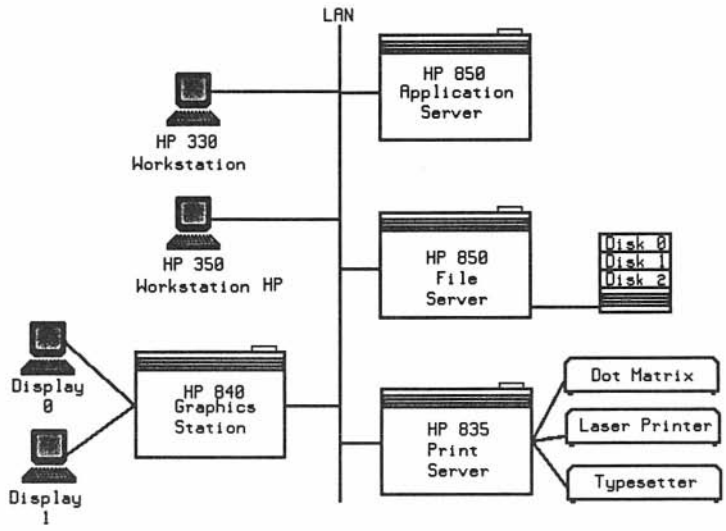


Figure 2-5. A Typical Distributed Computing Environment.

As the figure indicates, if you use a system running X11 and connected to a LAN, you have a multitude of resources available. The following sections provide a practical example of how the above environment and the resources contained therein could be used.

Workstations Provide Local and Remote Processing

Two workstations are pictured in the figure, an HP 9000 Series 330 and an an HP 9000 Series 350.

Both workstations can use clients that reside either locally, on their own hard disks, or remotely, on the hard disk of another system, for example the Series 850 application server. The workstations illustrate the capability of a single system to operate either locally or remotely.

Application Servers Handle Process-Intensive Applications

One of the HP 850s shown in the figure is an **application server.** An application server is a computer that provides the processing power and memory necessary to run large, processor-intensive applications.

A typical user of such an application would be Hank who works for a large oil company. Hank is currently involved in the search for new oil resources in Alaska. Many variables are considered in the attempt to locate potential oil fields. Hank uses a simulation program that mathematically manipulates all of these variables to produce data that indicates the potential for a certain area. These computations require a tremendous amount of memory, disk space, and processor time.

With a distributed computing environment, Hank can sit at his desk and use his personal workstation to log into the HP 850 application server and enter the necessary data. The actual simulation program and the necessary data files reside on the HP 850. Hank runs the simulation using the processing power of the application server. He has the output directed to a window on his workstation while he is busy performing tasks locally in other windows until the necessary simulation information is available.

Hank is only one of many employees to take advantage of the processing power of the application server. Other employees in the same department or even in a different building can also log in and use the system.

File Servers Supply Data Storage

The other HP 850 shown in the figure is a **file server.** A file server is a computer that controls the storage and retrieval of data from hard disks. A file server means less storage space is required on an individual's local computer. It also provides a relatively inexpensive and quick backup facility.

Let's say Alex is a writer who is responsible for the content of several chapters of a large manual. She works at her desk using an HP 330 as a writer's station and at any given time is working on one of several different projects that total 10 to 15 megabytes of storage on a hard disk. Using the HP 850 file server, she could store her files on a master disk drive and check out the chapters she needs to work on. This leaves a backup of the files on the file server. The file

server can thus be used to maintain current backups by transferring updated files to it on a regular basis.

At any given time, Alex will only have a few chapters stored on her own disk drive; those chapters she is currently working on. If she finds that she needs a copy of another chapter that is not currently residing on her disk, she requires only moments to transfer a copy from the main disk.

Another use for a file server is to serve as a hub for diskless workstations. You can have a cluster of several diskless workstations connected to a single hub with a large disk. Each workstation, or **node,** needs a certain amount of individual space on the disk, but all nodes can share the system and application software, eliminating the need for local system storage and thus saving a considerable amount in overall storage requirements.

Print Servers Control the Printers

The HP 835 shown in the figure has several printers attached to it and acts as a **print server,** a computer that controls spooling and other printing operations. Page formatters and page composition programs reside on the print server and are invoked with the proper commands. When you need a document printed using a particular type of printer, you send it to the print server with the appropriate instructions, and the task is accomplished. This permits a large number of individuals from anywhere in the distributed computing environment to efficiently share printer resources.

If Alex needed a copy of a chapter quickly printed for an immediate review, all she would need to do is to instruct the 835 to print the chapter using the fast dot-matrix printer. If she needed a letter-quality copy of a document containing elaborate graphics, she would route the letter to the laser printer. For those manuals that need to be typeset, the print server can also drive a typesetter. Alex would again simply direct the appropriate command to the print server to cause the document to be run through the typesetter.

Graphics Station for Specialized Graphics Applications

Certain applications are designed to take advantage of graphics accelerators in order to speed up the presentation of graphics on the screen. Generally, engineers working with CAD (Computer-Aided Design) applications are the major users of graphics accelerators. Hewlett-Packard supports graphics acceleration with a graphics library called Starbase. The HP 840 shown in the figure has two high-performance graphics subsystems attached to it. Each subsystem is powerful enough to run the X Window System in the display's overlay planes while running a Starbase application in the display's image planes.

Anne is an engineer who is working on a project involving the design of a new, high-speed sailboat hull. The CAD program she uses is very expensive and requires graphics acceleration to accomplish sophisticated shading. When Anne wants to work with the program, she can move to the graphics station where she can use multiple windows provided by X with the CAD program running in one of the windows.

The graphics station permits a larger number of people to share the expensive hardware and software resources required by a CAD/CAM station. Tasks that engineers may have that do not require graphics acceleration can be accomplished at their desks on a more typical workstation.

Multi-Vendor Communications

Another advantage of DCE is its ability to allow computers manufactured by different vendors, running different operating systems, to communicate with each other over the LAN. If you are using a computer made by Hewlett-Packard, you can communicate over the LAN directly with a computer made by Sun, DEC, IBM, or a variety of other manufacturers supporting X, as long as each is running the X Window System and connected to a LAN using the Ethernet protocol standard.

The diagram below shows a multi-vendor environment of computers running different operating systems. Communication over the LAN is a simple task as long as they are all running X11.

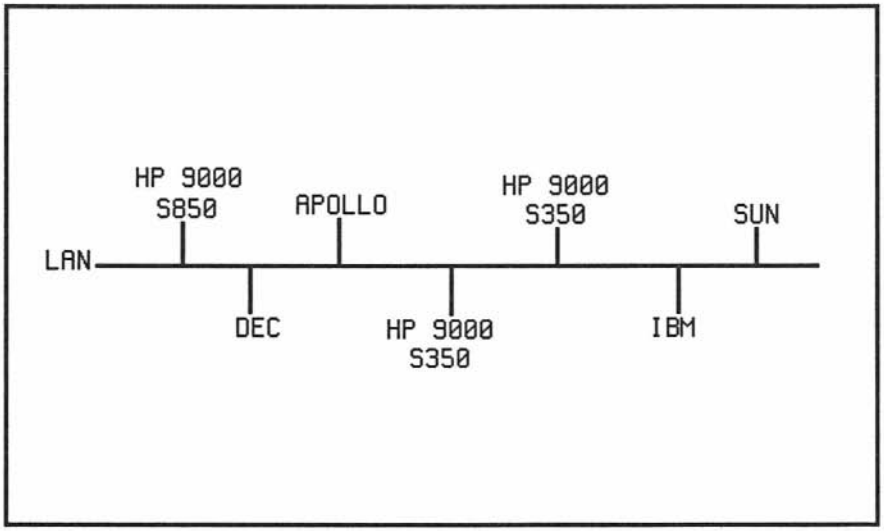


Figure 2-6. Multi-Vendor Communication Is A Benefit of X11 DCE.

Where to Go Next

You should continue to chapter 3 to learn how to use your X Window System. Chapter 4 contains information about running client programs from the command line, while the chapters following contain information on customizing your window system environment.

3

Using the X Window System

This chapter covers the basics of window operation. It shows you how to use X once it's been installed on your system. You'll learn how to perform the following tasks:

- Start the X Window System.
- Create, move, resize, and "shuffle" windows.
- Iconify a window and normalize an icon.
- Display menus and make selections.
- Stop programs and correctly exit your X environment.

Starting the X Window System

Before you start the X Window System, you must be logged in to your computer system. Log in using your normal procedure.

The X Window System can't run on a system that's *already* running HP Windows/9000. If you are running HP Windows/9000, you must exit from that window system *before* you start X. (HP Windows/9000 can be installed on your system; it just can't be running when you start X.)

Your system may be configured to start X11 as part of the login procedure. If so, skip the rest of this section and the next and start reading at "What to Expect When X Starts."

If your system is not configured to start X11 at login, log into the system in the usual way and type the following command at the command prompt:

```
x11start Return
```

You should start the X Window System just once. With X11 running, you should *not* execute the `x11start` command again. Starting X11 and then starting it again while it is still running may cause undesirable results.

Note, however, that you can restart the *window manager* and refresh the *screen* at any time.

Command-Line Options for x11start

In most cases, you will find it convenient to establish environment options in configuration files in your home directory. However, if you don't start X11 automatically at login, you can include environment options on the command line after the `x11start` command. The syntax for this is:

`x11start` [*−clientoptions*] `--` [*/server* [*−options*] / *:display* [*−options*]]

Client Options

Client options pass from the `x11start` command line to all clients in the `.x11start` file that have a `$@` parameter. The options replace the parameter. This method is most often used to specify a display other than the usual one on which to display the client. You can, however, use the command-line option to specify a non-default parameter, such as a different background color, for clients.

Server options

Server options are preceded with a double hyphen (−−). If the option following the double hyphen begins with a slash (/), it starts a server other than the default server. If the option begins with a colon followed by a digit (:#), it specifies the display number (0 is the default display number). Additional options specified after the server or display refer to the specified server or display. See the `Xserver` page in the reference section for more information on server options.

Examples

The examples below illustrate starting the X Window System in different ways.

`xllstart`	*The usual way to start X.*
`x11start -bg Blue`	*Gives clients followed by* **$0** *a blue background.*
`x11start -- /X2`	*Starts server X2 rather than the default server.*

Starting X on a Multi-Seat System

A multi-seat system (a system with more than one display, keyboard, and mouse) requires modification of two X11 configuration files, to allow for more than one display seat. These files, `X*screens` and `X*devices` (where * is the number of the display), are located in `/usr/lib/X11`. Each seat must have its own `X*screens` and `X*devices` files. If you have a multi-seat system but have not configured it, see your system installation or configuration manual for more information. Also see "Defining Your Display" in chapter 7.

Starting Seat 0

To start X11 on seat 0 (display 0) of a multi-seat system, log in as usual and type:

```
x11start Return
```

Seat 0 uses the `/usr/lib/X11/X0screens` and `/usr/lib/X11/X0devices` files to configure its output and input devices. These files are supplied with the system, but you must still match them to your hardware configuration.

Starting Seat 1

To start X11 on seat 1 (display 1) of a multi-seat system, log in as usual and type:

```
x11start -- :1 Return
```

Here the `--` signifies starting the default server while the `:1` specifies sending the output to seat 1. Seat 1 uses the `/usr/lib/X11/X1screens` and `/usr/lib/X11/X1devices` files to configure its output and input devices. If your system has a multi-seat configuration, you must create these configuration files using the `X0screens` and `X0devices` files as models.

What to Expect When X Starts

No matter how you start the X Window System, from the command line or automatically from a login file, when X starts, it always executes the same sequence of steps.

- It looks in your home directory for a `.Xdefaults` configuration file to read. If it doesn't find one, it reads `/usr/lib/X11/sys.Xdefaults` instead.
- If necessary, it adds the path to X11 programs (`/usr/bin/X11`) to your `PATH` statement.
- It looks in your home directory for a `.x11start` configuration file to read. If it doesn't fine one, it reads `usr/lib/X11/sys.x11start` instead.
- It starts `xinit`, which starts the server and any clients specified in the `.x11start` configuration file.

You won't notice any effect from issuing the command until the X display server starts.

The Server Starts the Root Window

When `x11start` starts the server (the program that controls the operation of your keyboard, mouse, and display), your screen will turn gray. This means that the screen has now become the **root window,** the backdrop or "desktop" on which the windows and icons of your environment appear. Although you can completely cover the root window with clients, you can never cover a client with the root window. The root window is *always* the backdrop of your window environment; nothing gets behind it.

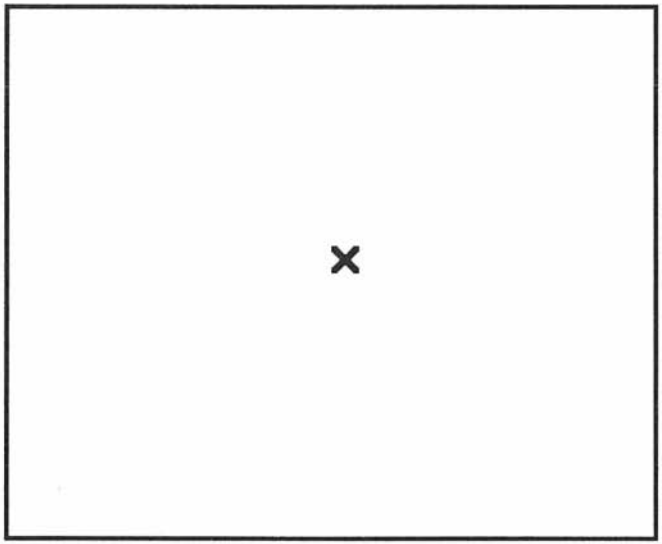


Figure 3-1. The Screen Becomes the Root Window.

In the center of the root window is an ×. This is the **pointer** and marks the current screen location of the mouse.

A Terminal Window Appears on the Root Window

A short time later a terminal window appears at the top of your display (if you're using the default `.x11start` file). This window is under the control of a window manager. If you use the `uwm` window manager, the window appears as an unframed rectangle. If you use the HP Window Manager, `hpwm`, you will notice that your window has a three-dimensional frame. This frame contains window manager controls.

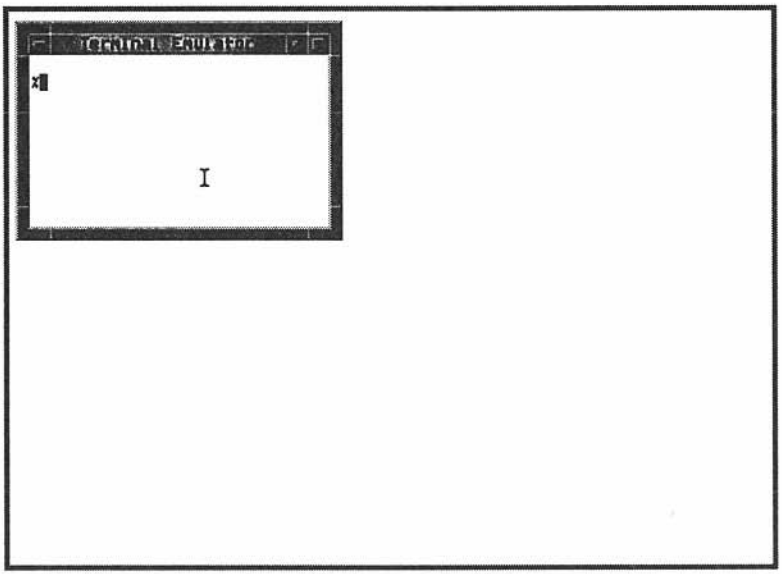


Figure 3-2. The Default X Environment: 'hpwm' and One Window.

The window that `x11start` creates is an X11 client called `hpterm` and is called an **hpterm window** to distinguish it from other types of window clients. The window contains a command-line prompt and behaves exactly like the screen of a standard HP terminal. You can think of this window as "a terminal in a window."

Move the mouse. The pointer moves on the screen. When the pointer is in the root window, it has an × shape. However, when you move the pointer to a terminal window, the pointer changes to an arrowhead (when on the window frame) or an `I` (when in the interior of the window).

With the HP Window Manager, when you press and release the select button while the pointer is in a terminal window, the window becomes the **active window.** When a window is active, its frame changes color. You'll discover that you can't type in a terminal window unless the window is active.

The active window is the terminal window where what you type on the keyboard appears. *Your input always goes to the active window.*

If there is no active window, what you type is lost.

The program running in the active window decides what to do with your typed input. Frequently the program will use a **text cursor** to show where your typed input will be displayed.

What to Do If X11 Doesn't Start

Table 3-1. Possible X Window System Start Problems.

If this happens ...	You should do this ...
The message `command not found` appears.	Check your spelling and reenter the start command.
The root window displays for a moment, but then goes blank.	Press the Return key to bring back your original command-line prompt and see below.
The root window displays, but no pointer appears.	Press CTRL Left Shift Reset all at the same time. This brings your original command-line prompt back. See below.
The root window and pointer display, but no terminal window appears.	Press *and hold* the menu button. If a menu appears, open a window. Otherwise, press CTRL Shift Reset and try restarting X, then see below.
The terminal window displays, but what you type doesn't appear after the window's command prompt.	Move the pointer into the window and click (press and release) the select button, then type.

If you encounter problems starting X11 for the first time, check the following areas:

- Check the X11 start log in your home directory for clues by typing `more .x11startlog` Return.
- Check your system's `PATH` statement by typing `env` Return. The `PATH` should begin with `/usr/bin/X11:`.
- Check that the `DISPLAY` environment variable (type `env` Return) is set either to `local:0.0` or *host:0.0* where *host* is the hostname of your system.

- Check the `.x11start` file in your home directory for errors. Compare it with the `/usr/lib/X11/sys.x11start` file.

If none of the above seems to help, or you're not sure how to proceed, see your system administrator.

Working With Windows

In the typical X environment, you have two tools to control window operations:

- The mouse.
- The window manager.

For most window operations, you'll use a combination of the window manager and mouse. (If you lack the space on your desktop, or feel more comfortable with a keyboard, you can configure your keyboard to take the place of the mouse.)

Which Mouse Button Does What

The X Window System works with either a two-button mouse or a three-button mouse. If you have a two-button mouse, you can emulate a three-button mouse. The following table explains which button is which.

Table 3-2. Which Mouse Button Is Which.

To press this ...	On a 3-button mouse press ...	On a 2-button mouse press ...
Select Button	the left button	the left button
Alternate Button	the middle button	both buttons
Menu Button	the right button	the right button

Besides using the mouse to point with, you use the mouse buttons to select an operation to be performed on the object pointed to. Buttons have the following actions associated with them:

Press	To hold down a button.
Click	To press *and release* a button without moving the pointer.
Double-click	To click a button twice in rapid succession.
Drag	To press *and hold down* a button while moving the pointer.

The Anatomy of an hpwm Window Frame

The HP Window Manager surrounds each window on the root window with a functional frame. Positioning the pointer on a part of the frame and performing a mouse button action will execute the function of that part of the frame.

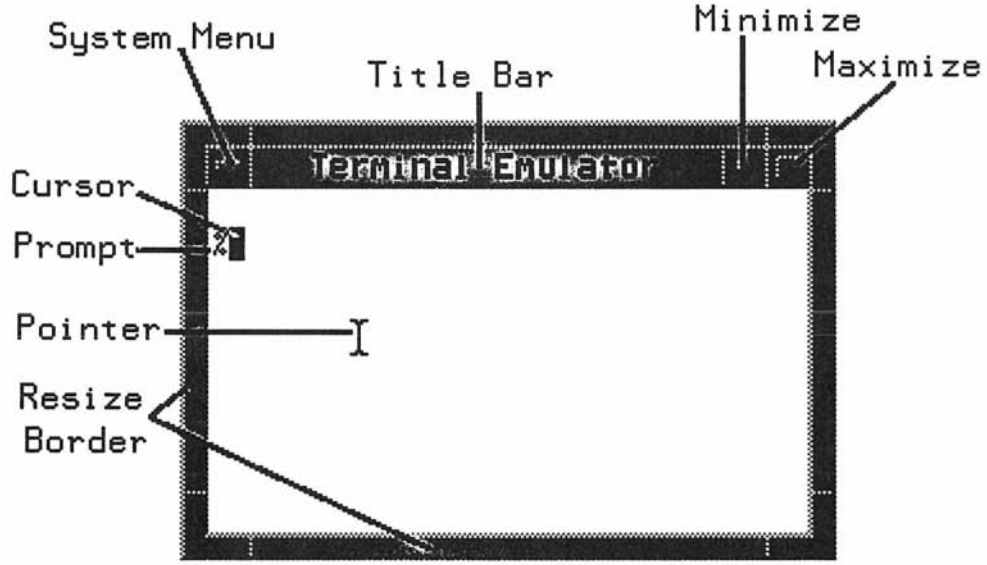


Figure 3-3. The HP Window Manager Surrounds a Window with a Frame.

The parts of the `hpwm` window manager, their functions, and the required mouse operations are listed in the following table.

Table 3-3. Window Frame Parts and What They Do.

Frame Part	Function	Mouse Action
Title bar.	Move a window.	Press and drag the select button.
System menu button.	Display a system menu.	Press and drag the select button.
Minimize button.	Iconify a window.	Press the select button.
Maximize button.	Expand window to maximum size.	Press the select button.
Corner pieces.	Stretch or shrink a window diagonally (in two directions).	Press and drag the select button.
Frame sides.	Stretch or shrink a window horizontally.	Press and drag the select button.
Frame top and bottom.	Stretch or shrink a window vertically.	Press and drag the select button.

Displaying and Selecting from the System Menu

Every window has a system menu. The system menu button of a window is in the upper left corner of the window frame next to the title bar. You can display the system menu at any time by pressing the select button with the mouse pointer on the system menu button.

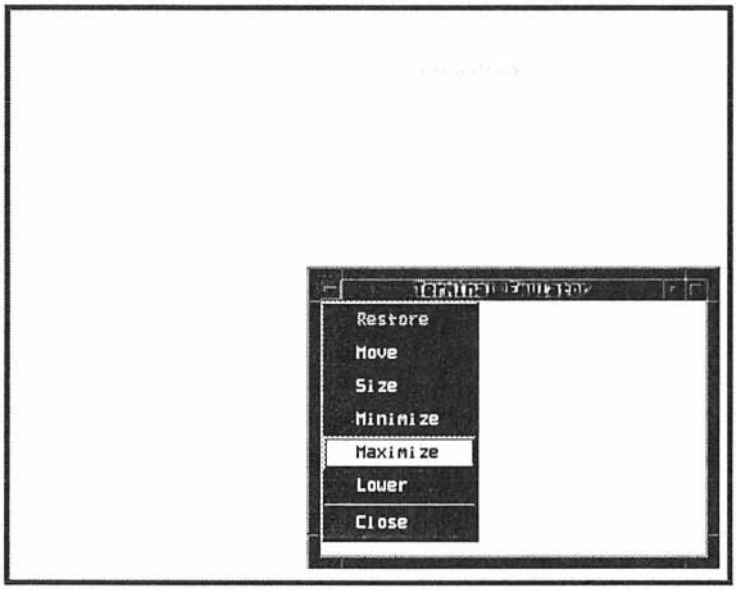


Figure 3-4. Every Window Has a System Menu.

To display a window's system menu and make a selection, do the following:

1. Position the pointer on the system menu button.
2. Press *and hold down* the select button.
3. Drag the pointer down the menu to the selection you want to choose.
4. When the selection highlights, release the select button.
5. (Move and Size only.) Move the pointer to the desired location or until the desired size is achieved, then click the select button to end the operation.

If you change your mind and don't want to make a selection, move the pointer off the menu area *before* you release the select button.

You can also display the system menu by pressing Left Shift ESC. To make a choice using this method, use the ▲ and ▼ keys to highlight a selection, then press Return. If you don't want to make a selection, press Left Shift ESC again.

The table below describes the system menu selections.

Table 3-4. The System Menu Selections.

To do this ...	Select ...
Restore a window from an icon or after maximizing.	Restore
Change the location of a window.	Move
Change the width and height of a window.	Size
Shrink a window to its icon (graphic representation).	Minimize
Enlarge a window to cover the entire root window.	Maximize
Send a window to the back or bottom of the window stack, the position closest to the root window.	Lower
Immediately stop the window and make it disappear.	Close

The rest of this chapter explains how you can use the mouse and the window manager to control the windows in your environment.

Moving a Window around the Screen

You can move any window (except the root window) by doing the following:

1. Position the mouse pointer in the title bar.
2. Grab the title bar by pressing *and holding down* the select button.
3. Drag the pointer. An outline of the window shows you the window's new location.
4. Position the outline and release the select button to relocate the window.

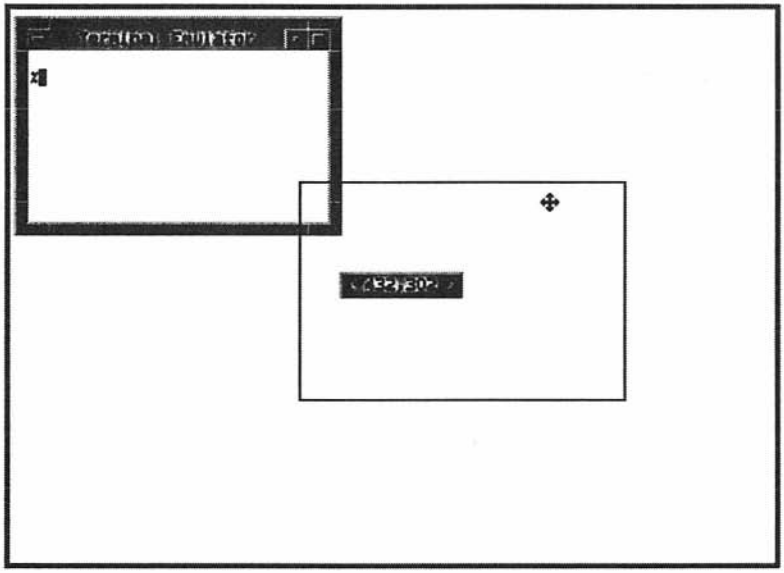


Figure 3-5. An Outline Shows the Window's Location.

You will notice that, along with the window outline, a small location box displays at the center of the screen. The numbers in this box are the column and row position of the upper left corner of the actual window (the area inside the window frame). The measurement is in **pixels.** Pixels (short for picture elements) are tiny dots, arranged in rows and columns on the screen, that make up the images that display.

As mentioned in the previous section, you can also move a window by choosing the "Move" selection from the system menu.

Changing the Size of a Window

To change the size of a window, grab the window's frame with the pointer, drag the frame to the desired size, and then release the frame.

Where you grab the frame determines how the window gets resized. If you grab the side of the frame, the window stretches or shrinks horizontally. If you grab the top or bottom of the frame, the window stretches or shrinks vertically. If you grab the frame by one of the corner pieces, you can expand or contract the size of the window in two directions at once.

Table 3-5. Where to Grab a Window Frame.

If you want to stretch or shrink the window ...	Position the pointer on the ...
vertically from the ...	
top	top of the frame, above the title bar
bottom	bottom of the frame
horizontally from the ...	
right	right side of the frame
left	left side of the frame
diagonally from the ...	
bottom left corner	frame's lower left corner
top left	frame's upper left corner
top right	frame's upper right corner
bottom left	frame's lower right corner

The pointer changes shape when you're positioned correctly for the grab.

Follow these steps to grab and resize the window:

1. Position the mouse pointer on a part of the window frame.
2. Press *and hold* the select button.
3. Drag the mouse pointer. An elastic outline represents the new window size.
4. Release the select button when the elastic outline is the correct size.

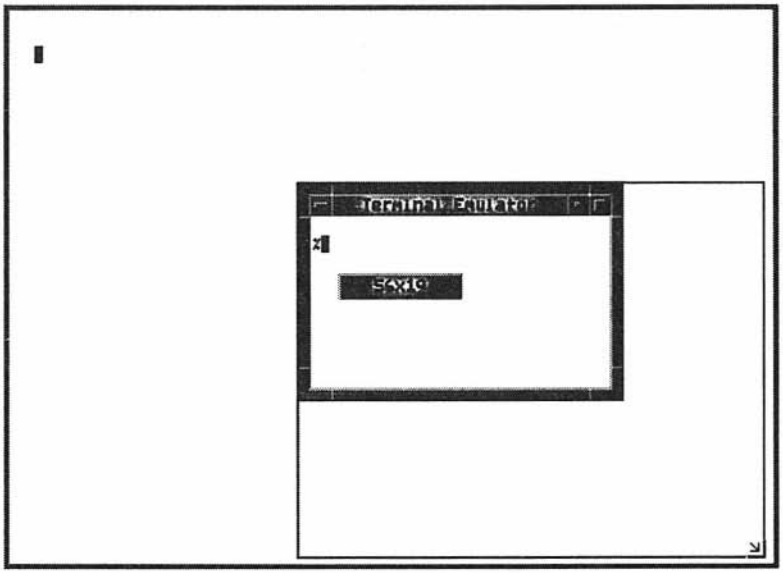


Figure 3-6. An Elastic Outline Shows the Window Size.

Although you change a window's size and shape during a resize operation, you do not change its position. The section of the frame opposite where you grab always remains in the same location.

As mentioned earlier, you can also resize a window by choosing the "Size" selection from the window's system menu. If you choose the "Size" selection, you must cross the window frame's border with the pointer before the elastic outline appears.

Raising a Window to the Top of the Window Stack

As you open more and more windows during a work session, your screen will become cluttered as some windows become obscured under other windows. The windows appear "stacked" on top of one another.

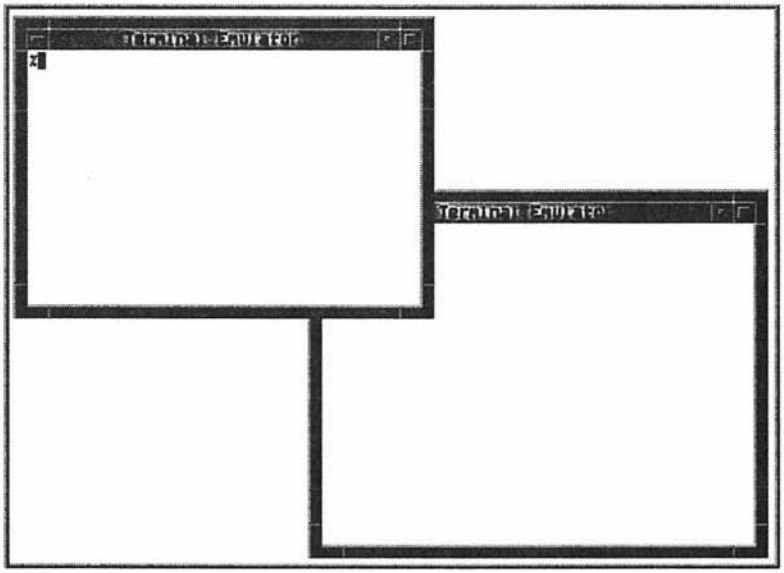


Figure 3-7. Windows Become Obscured by Other Windows.

To raise a window to the top of the stack (front of the screen), position the pointer on any visible piece of the obscured window's frame and click the select button. This also makes the window the active window.

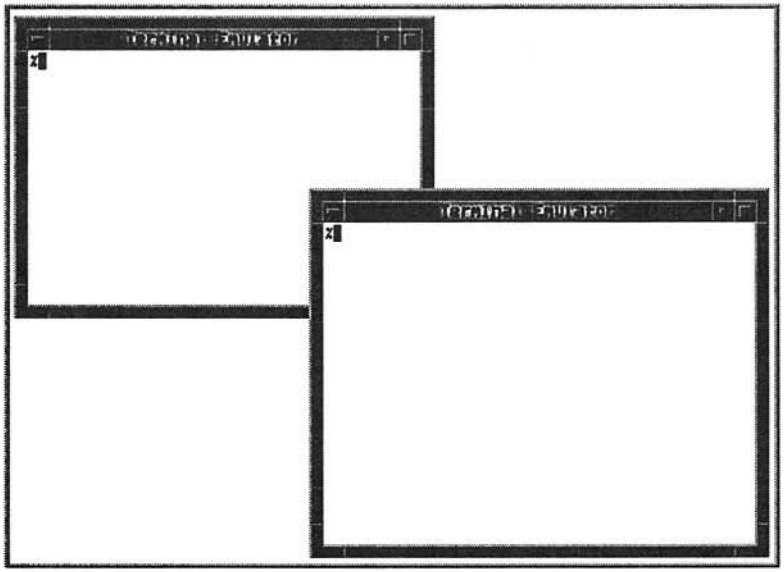


Figure 3-8. A Window Is Unobscured by Raising It.

An alternative in some situations is to lower the window on top of the stack by choosing the "Lower" selection from that window's system menu.

Iconifying a Window

Sometimes raising a window isn't enough to solve the problem of a cluttered root window. You can save space and bring order to your workspace by reducing inactive windows to icons—small, easily-recognizable graphic images that represent full-sized windows. Later, as you need them, you can change the icons back into full-sized windows.

Figure 3-9. Pressing the Minimize Button Iconifies a Window.

Changing a window into an icon is known as **iconifying** or **minimizing** the window. To iconify a window:

1. Move the pointer to the minimize button located in the upper right corner of the window frame between the title bar and the Maximize button.
2. Press *and release* the select button.

Immediately after you release the select button, the window is iconified. Successive icons are placed from left to right in a row along the bottom of the root window using a grid pattern. This placement is by default and can be changed if your needs require it.

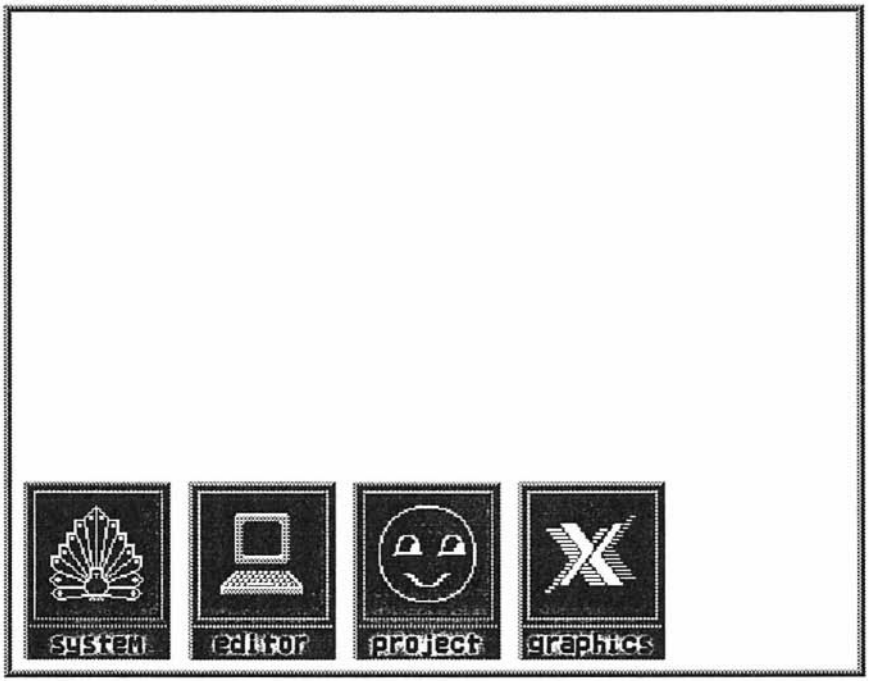


Figure 3-10. Default Icon Placement Is along the Screen's Bottom.

You can also change a window into an icon by choosing the "Minimize" selection of the system menu as discussed earlier under the system menu.

Turning an Icon Back into a Window

When you have room on the root window, or simply want to check the progress of an application running in an iconified window, you can turn the icon back into a window. Changing an icon into a window is called **normalizing** or **restoring.**

1. Move the pointer to the icon.
2. Double-click the select button (press *and release* it twice in rapid succession).

After you double-click on the icon, the window will reappear located at its previous (pre-iconified) position.

More Work with Icons

Although you can't enter information into an icon, any program running in a window as it is iconified continues uninterrupted until it either completes or pauses to await input from you.

Icons allow you a way to start an application in a window and then collapse the window into a tiny symbol over in the corner of your screen. There the program quietly does its work without cluttering up your workspace.

Displaying and Selecting from an Icon's Menu

Although an icon doesn't have a frame like a window, it does have a system menu that gives you the standard control options with the exception of "Size" and "Minimize," which appear on the menu but don't function with iconified windows.

To display an icon's system menu and make a selection:

1. Move the mouse pointer over the icon.
2. Press *and hold down* Shift, and press ESC to display the menu.
3. Use the ▲ and ▼ arrow keys to highlight the proper selection.
4. Press Return to make your selection.

To make no selection, press *and hold down* Shift, and press ESC. The menu will disappear.

Moving Icons around the Screen

Although icons appear by default in a row along the bottom of the screen, you can move them anywhere on the root window.

To move an icon:

1. Move the mouse pointer onto the icon.
2. Press *and hold* the select button.
3. Drag the pointer to a new location. An outline of the icon shows the current location.
4. Release the select button.

Displaying and Selecting from the Root Menu

The root window has its own menu called (not surprisingly) the **root menu.** You can display the root menu any time the mouse pointer is on the root window. When the pointer is in the root window, remember, it has an × shape.

To display and select from the root menu:

1. Position the pointer anywhere in the root window.
2. Press *and hold* the menu button to display the menu.
3. Drag the pointer down the menu until you have highlighted the desired selection.
4. Release the menu button.

To make no selection, move the pointer off the menu *before* you release the menu button.

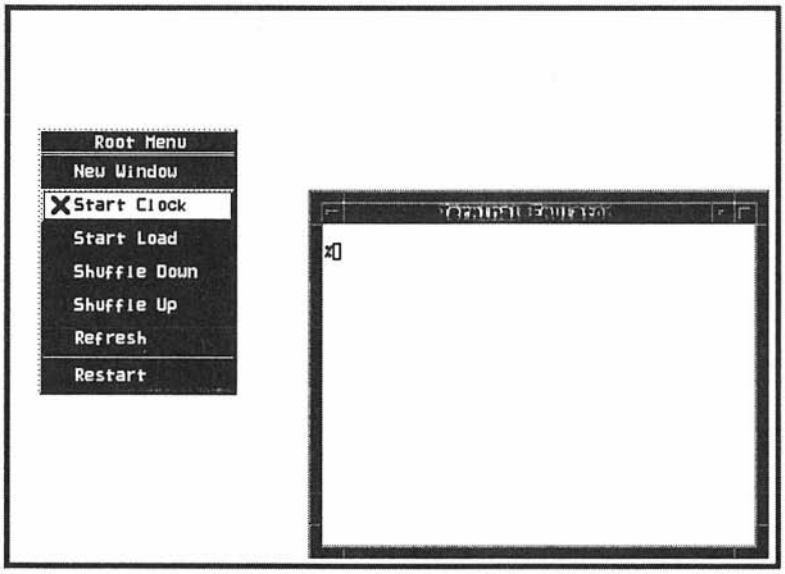


Figure 3-11. The Root Menu Provides Screen-Wide Functions.

The default selections of the root menu provide you with screen-wide functions not appropriate for an individual window's system menu.

Table 3-6. What the Root Menu Default Selections Do.

To do this ...	Choose this selection ...
Make a new 80×24 **hpterm** terminal window near center screen.	New Window
Display an analog clock in the upper right corner of the root window.	Start Clock
Display a histogram measuring system load (displays next to the clock).	Start Load
Bring the most concealed window to the front of the window stack.	Shuffle Up
Lower the least concealed window to the bottom of the window stack.	Shuffle Down
Blank out then redisplay the screen (useful if video images become corrupt).	Refresh
Restart window manager to see recent configuration changes.	Restart

Exiting From the X Window System

Exiting from the X Window System means stopping the X11 display server. Leaving X places you back at the command prompt you had immediately after you logged into your system.

Before stopping the X Window System, you must first stop any application programs you may have running. This ensures that you do not unknowingly leave any orphaned processes executing. It also ensures that all open files are properly closed to prevent loss of data.

Caution

Stop all application programs before stopping the window system. If you don't do this, any open files may not be updated properly. This could result in the loss of valuable data.

Stopping Application Programs

You can stop a program and remove its window in two ways.

Following the Program's Normal Exit Procedure

The best way to exit a program is to use the program's usual "exit" procedure. This should always be your preferred method for stopping the program. Many programs have commands or keystrokes that stop them.

If the program is a client and created its own window, the window is removed when the client stops. If the program is a non-client in a terminal window, the window remains, and you can stop it when you stop the display server.

Stopping the Window System

After stopping all application programs, stop the window system by holding down the CTRL and left Shift keys, and then pressing the Reset key. This stops the display server, and with it the window system.

What Next

Now that you've experienced the X Window System and learned how to control your terminal window, you're ready to use X as your working environment.

Chapter 4 contains information about the viewable clients supplied with the X Window System and how to run them from the command-line of a window. Chapter 5 describes how you can incorporate these clients into your environment.

Successive chapters supply increasingly more detailed information about the HP Window Manager and other "non-viewable" clients.

4

Running from the Command Line

You can divide the programs you run in your X environment into two groups:

clients Programs written specifically to take advantage of the windowing capability of the X Window System. Clients are the tools you use to work in your X environment.

non-clients Programs written for terminals, not window systems. You can run a non-client in the X Window System by creating a terminal emulation window in which to run the non-client.

This manual uses clients to mean "window-smart" applications, non-clients to mean "terminal-based" applications, and "programs" to refer to both clients and non-clients.

You will probably start the programs that you use frequently either automatically, as part of your X environment, or by choosing them from a menu. However, you can start any client from a command-line prompt.

This chapter discusses the following topics:

- X11 clients and what they do.
- Command-line syntax.
- Starting programs from the command line.
- Stopping programs.
- The `hpterm` terminal emulation client.
- The `xterm` terminal emulation client.
- The `xclock` client.
- The `xload` client.
- Working with common client options.

- Troubleshooting command-line programs.

Meeting the X11 Clients

This chapter discusses four clients. Other clients are discussed in the following chapters as the functions they control are discussed. But to give you an idea of the tools that are available in the X environment, this section gives you a brief overview of X11 clients and client options.

What the X11 Clients Do

The following tables group the X11 clients (somewhat artificially) into functional categories and give you a brief idea of what the clients do:

Table 4-1. X11 Clients That Initialize and Configure.

To do this ...	Use this client ...
Initialize the X Window System and start the server.	`xinit`
Provide a display with the X communication protocol.	`xserver`
Start `xinit`, X, and X clients.	`x11start`
Alter the modifier-key mappings of a keyboard.	`xmodmap`
Adjust display preference options.	`xset`
Initialize a new colormap for an X environment.	`xinitcolormap`
Create a color database for X.	`rgb`
Add a new remote host to your system.	`xhost`
Load a window manager's resource configuration into the server.	`xrdb`
Compile a BDF-formatted font into an X server format.	`xfc`

Table 4-2. X11 Clients That Control Window Management.

To do this ...	Use this client ...
Resize the contents of a window.	`resize`
Repaint the display screen.	`xrefresh`
Find out information about windows.	`xwininfo`
Provide window manager services to your environment.	`uwm`
Provide window manager services to your environment.	`hpwm`

Table 4-3. X11 Clients That Control Graphics Functions.

To do this ...	Use this client ...
Open a window into a graphics workstation overlay plane.	`xseethru`
Make a screen dump (bitmap).	`xwd`
Translate an **xwd** bitmap to Starbase format.	`xwd2sb`
Translate a Starbase bitmap to **xwd** format.	`sb2xwd`
Print a screen dump on a PCL-format printer.	`xpr`
Stop multiple Starbase X windows.	`gwindstop`
Create a new X window for Starbase.	`xwcreate`
Destroy a Starbase X window.	`xwdestroy`
Display a previously made screen dump.	`xwud`

Table 4-4. X11 Clients That Provide Viewable Services.

To do this ...	Use this client ...
Make a window that emulates an HP terminal.	`hpterm`
Make a window that emulates a DEC or Tektronix terminal.	`xterm`
Display a clock telling the system time.	`xclock`
Display a histogram telling the system load.	`xload`
Make a bitmap for a cursor, icon, or root window tile.	`bitmap`
Display the characters of a particular X font.	`xfd`
Set the color and appearance of the root window.	`xsetroot`

If your interest is in running applications in the X environment, you probably won't ever use some of the clients listed above. If your primary interest is in programming, graphics, or the more technical aspects of environmental control, chapters 6 through 8 and the man pages are your definitive source of information.

Specifying the General Syntax for Command-Line Starts

Starting clients from the command line of a terminal window gives you a way to dynamically alter the elements that compose your X environment. To start a client from a command line, you must have X11 running, and you must use the correct command-line syntax.

Specifying the Syntax

The general syntax for all clients that you start from a command line is the same:

client [*−options*] [&] Return

Options enable you to control the appearance and behavior of a client that you start from a command line. Each client has its own options, but some clients, such as the viewable clients discussed later in this chapter, use the same options. The reference section contains the complete list of all client options.

You specify an option after the client name. The option begins with a hyphen (−) and includes the option itself and an argument. For example, the following is a typical command line to start an `hpterm` window with a black background and white foreground:

```
hpterm  -bg  Black  -fg  White & Return
```

Choosing Background Processing

An important element of the command-line syntax is the ampersand (`&`) which ends the command line. As mentioned earlier, the `&` is what tells the system to start the client as a background process, a process that doesn't require the total attention of the computer. Background processing enables you to have more than one client running at the same time and frees your keyboard for further use.

Although the `&` is an optional element, and you can choose to run a client as a foreground process if you desire, you will probably find that in most cases, you will use background processing.

Starting Programs

You can start a client either locally or remotely. A **local client** is a program that is running on your "local" system, the same system that is running your X server. A **remote client** is a program that you view from your local display, but the program actually resides and is running on a system other than yours, a "remote" system.

Starting Local Clients

You can start a local client from the command line any time after you've started X11 and have a window displayed which has a command prompt. To start the client, type the name of that client, followed by any options, then press Return.

It isn't necessary to specify options to run the client; just typing the client name and pressing Return will start the client using a list of option default values. System-wide defaults are contained in the

`/usr/lib/X11/sys.Xdefaults` file. Options that override these system-wide defaults are contained in the `.Xdefaults` file in your home directory. Command-line options, as you might suspect, override both of these default files.

For example, the following gives you the default clock client: an analog clock updated every 60 seconds:

```
xclock &  Return
```

You can, however, override these defaults and start a clock client with a digital readout in the lower left corner of the screen.

```
xclock -digital -geometry 160x25+1-1 &  Return
```

Starting Local Non-Clients

A non-client normally relies on a terminal instead of a window for displaying its output. To start a non-client program in an X11 window environment, you must first create a terminal emulation window, and then run the non-client in that window.

The following example simply creates an `hpterm` window. Using the command prompt in the window, you can operate most HP-UX system commands (the exception being a command like `update` which affects the entire system, not just the X environment).

```
hpterm &  Return
```

The window opens with its command prompt in the same directory as its **parent window,** the window from which it was started.

At any command-line prompt in any X window, you can start a non-client program simply by typing the start command for that program (usually the program's name) followed by a Return. For example, you could type the following at the command-line prompt:

```
banner windows are great  Return
```

The command prints a banner on the window.

Starting Remote Clients

A remote client is an X11 program running on a computer that is *not the same* computer that the X server is running and displaying on. In other words, the hallmark of a remote client is that the client runs on one computer while the output displays on another.

You can start a remote client from the command line any time after you've started the X Window System and have a window with a command prompt. You can start a client on any **remote host** to which your system has access. A remote host is the computer system that runs the remote client.

Gaining Remote Access

To gain access to a remote host, you must meet all of the following criteria:

- Be on a network with other systems. (This manual uses the *NS-ARPA Services* commands in all examples.)
- Have the internet address and hostname of the remote host in your system's `/etc/hosts` file.
- Have a valid login on the remote host.
- Have the remote host listed in the `/etc/X0.hosts` file.
- Have the remote host listed a `.rhosts` file in your home directory on your local system. (You may also want to have your local system listed in a `.rhost` file *on the remote host.*)

The first three criteria provide basic network capability to your system. You must have them to use the network whether or not you use the X Window System. The last two criteria provide your local X server with the ability to use the network. The **.rhosts** file lists the systems that have permission to use your username and account to access a system without formally logging in. The **X0.hosts** file contains a list of all X11 hosts known to your X server. The "0" signifies that the file is used by display 0 (similarly, display 1 would use an `X1.hosts` file).

Note

A `.rhost` file allows someone to access your login account *without giving a password.* Depending on your situation, this may pose a threat to the security of your system or the network your system is on. Check with your system administrator and carefully analyze your seurity needs.

Starting the Client

You have two choices when it comes to running clients on a remote host:

- You can log into the remote host and run a client.
- You can start a client remotely without formally logging in.

In either case, you need to select the display on which you want the output to appear.

Selecting the Display

Just as you need to select a remote host on which to run a client, so too you need to select a display on which the client's output appears. Typically this will be the display attached to your system, but it doesn't have to be.

For example, you could be sitting at your system reviewing lab reports kept on a (remote) lab system when you get the idea to show the reports to Turner at another division. You call to make sure Turner is in, then open a window on Turner's system, display the lab report that interested you, and discuss its significance with Turner without the delay or trouble of making a physical copy of the report and mailing it.

To help you in selecting a display, viewable clients have a `-display` option that allows you to specify on the command line which system is to receive the output. The syntax for the option is as follows:

`-display` *host*:*display*.*screen*

The *host* specifies the hostname of the system where you want the remote client's output to appear (usually your own system). The *display* is the number of the display where the output is to appear (usually 0 on an HP S300 and 0-3 on an S800). The *screen* is the number of the physical screen were the output is to appear (usually 0).

Examples of Starting Remote Clients

The following examples illustrate several ways of doing the same thing: starting an **xload** client on remote host **hpcvfaa** and displaying it on the console of your local system **hpcvfbb**.

Example 1: Logging In to a Remote Host the Wrong Way. At the command-line prompt of an existing terminal window, you could type the following:

```
rlogin  hpcvfaa  Return
xload  -display  hpcvfbb:0.0  Return
```

Using this command is a mistake in most cases. Note the **&** is missing from the end of the command line. This command would not return a command prompt to the window until you stopped the **xload** client. Your window would effectively be "frozen."

Example 2: Logging In before Running the Client in Background. At the command-line prompt of an existing window, you could type the following:

```
rlogin  hpcvfa  Return
xload  -display  hpcvfb:0.0  &  Return
```

Similar to example 1, these two command lines log you in and then start the xload client, this time as a background process. This leaves your original window free for use. The display is again to your system's console.

Example 3: Using a Remote Shell to Start a Client. At the command-line prompt of an existing window, you could type the following:

```
remsh hpcvfa -n /usr/bin/X11/xload -display hpcvfb:0.0 &  Return
```

Respectively, this command starts a remote shell, on remote host **hpcvfa**, redirects **remsh** input (necessary in this case), starts the client **xload**, and directs output to system **hpcvfb**, display 0, screen 0, as a background process.

Note that you wisely used the full path to the **xload** client when starting it. This is a good idea especially in situations where the remote machine might have two versions of the same client (for example, an X10 and an X11 version of **xload**).

The benefit of using a remote shell instead of a remote login is that, with a remote login, the local system starts two processes (the remote login and the client), while with a remote shell, the local system starts only one.

Starting Remote Non-Clients

Starting a remote non-client is similar to starting a remote client except that, before you start the non-client, you must first start a terminal emulation window in which to run the non-client.

Starting the Non-Client

You can always log into the remote host and start a non-client. Using an existing window essentially makes that window a "terminal" of the remote host. Output from the non-client appears in the window. When you exit the non-client and the remote host, the window "returns" to the local system.

Starting a non-client using a remote shell such as `remsh`, however, is sometimes inappropriate. To use a remote shell, you must first create a terminal emulation window in which to run the non-client. If the non-client executes too quickly, you may not see the results, since, once the non-client finishes executing, the emulation window to the remote host closes.

Table 4-5. Choosing a Method of Displaying Remote Processes.

If you want the window to ...	Do this ...
Remain after you have finished the initial remote process.	Use an existing window to log in to the host before executing the remote command.
Disappear after you're finished with the remote process.	Execute the command as an option of creating a new window.

Example 1: Logging In to a Remote Host before Running the Non-Client. At the command-line prompt of an existing window, you could type the following:

```
rlogin hpcvfa Return
ll Return
```

If you are familiar with networks, you probably recognize this command. It simply logs you in to a remote host, `hpcvfa`, and then uses the HP-UX `ll`

command to list the files in your home directory on that host. Remember, operating system commands, because they are part of HP-UX and not the X Window System, are non-clients.

Example 2: Starting a Window That Starts a Remote Non-Client

At the command-line prompt of an existing window, you could type the following and press Return:

```
hpterm -display hpcvfb:0.0 -e remsh hpcvfa -n ll &
```

This example starts another `hpterm` terminal emulation window client. As the first option of that client (`-display`), the ouput is directed to your local display (`hpcvfb`). As the second option (`-e`), the `hpterm` client executes a remote shell on `hpcvfa` that connects the window to a remote host (hpcvfa) and lists the files in your home directory there.

Although, at first glance, this command line appears to do the same thing as example 1, there is an important difference. When the `ll` command of example 2 finishes executing, the window created for it to run in will disappear *whether or not you've had time to view all the files.* Therefore, this is a poor command syntax to use in this situation.

Example 3: Starting a Remote Non-Client Window. At the command-line prompt of an existing window, you could type the following and press Return:

```
hpterm -display hpcvfb:0.0 -e remsh hpcvfa -n vi report &
```

This example is the same as example 2 except that the non-client started is differend. You start `vi` and open the `report` file. In this case, the window stays displayed until you exit `vi`. You could edit `report` and exit, closing the window. Or you could save `report` and read in another file. As long as you didn't exit `vi`, your "remote editing window" would stay displayed.

Stopping Programs

How you stop a program you've started from a command line depends on whether the program is a client or non-client.

Stopping Clients

Clients like **xload** and **xclock** have no data to save. You stop them by choosing the "Close" selection from the system menu.

Other clients, like **hpterm**, **xterm**, and **bitmap**, may contain data you want to save. Save the data *before* you stop the client. In the case of terminal windows, a non-client running in the window may actually contain the data. Stop the non-client in the approved manner before you stop the window. When you have a command-line prompt in a terminal window, you can stop the window. In the case of **bitmap**, use the "Write Output" selection on the sidebar menu to save the bitmap before you stop the client.

After you have saved any data and exited any non-clients (in the case of terminal windows), stop the client by choosing the "Close" selection from the client's system menu. Note that if you started a non-client as an option of creating a window, when you stop the non-client, the window will stop.

Stopping Non-Clients

Stop all non-clients in the manner approved in the instructions for that non-client. Generally, a non-client program stops automatically when it finishes executing or has a "stop" provision.

Killing Programs That Won't Stop

If for some reason (and you will no doubt discover some) you cannot shut down a program in the normal manner, you should "kill" the program before you exit the window system. Killing the program means using the HP-UX **kill** command to stop the program's execution environment or "process."

Other Ways to Stop a Program

Before you use the `kill` command to stop a program's process, try the following key sequences:

- Press CTRL and, while holding it down, press c.
- Press CTRL and, while holding it down, press d.
- Press q.

Killing the Program's Process

If none of these key sequences stop the program, use the following steps to `kill` the program's process:

1. Save any data that needs saving.
2. Find the PID (process ID) for the program by typing the following:

   `ps -fu` *username* Return

   where *username* is your login name. The `ps -fu` command lists all the processes running under your login name. You should be able to identify the program you want to kill by looking for it under the "COMMAND" column (the rightmost column in the list). The PID for the program will be located in the second column from the left.
3. To kill the program, type:

   `kill -2` *pid* Return *The equivalent of* CTRL c.

   where *pid* is the PID number.
4. If this doesn't work (type another `ps -fu` command), type:

   `kill -3` *pid* Return *A stronger version of* `kill`.
5. If this still doesn't work, type:

   `kill -9` *pid* Return *The strongest version of* `kill`.

You can kill several programs at once by including several PIDs separated by spaces in the command. Just be careful that you have the correct PIDs.

Terminal Emulation Clients

The X Window System comes with the following two terminal emulation clients:

`hpterm`	Emulates a Term0 terminal.
`xterm`	Emulates DEC VT102 and Tektronix 4014 terminals.

Emulating an HP Terminal with the 'hpterm' Client

The `hpterm` terminal emulation window is the default terminal used by your X Window System and provides you with basic access to your system. The window's command-line prompt functions exactly like the command-line prompt of an HP **Term0** terminal. Term0 defines an HP level 0 terminal; it is a reference standard defining basic terminal features. For more information about Term0 terminals, see *Term0 Reference* in the HP-UX documentation set.

The `hpterm` window client includes the following features:

- Escape sequences that control terminal operation.
- Eight definable softkeys.
- Full Roman8 character set (ASCII and Roman Extension).
- Two character fonts (base and alternate).
- Screen editing functions.

If your needs require one or more of these features, see chapter 7, "Customizing Special X Environments" where they are discussed in detail.

Syntax

The syntax of the `hpterm` window client is as follows:

`hpterm` [*−options*] [&]

You'll find a list of common viewable-client options in "Working with Common Client Options" later in this chapter. For a complete list of `hpterm` options, see the `hpterm` pages in the Reference section.

Using 'hpterm' Terminal Window Softkeys

The `hpterm` client softkeys work exactly like an HP Term0 terminal's softkeys. To display `hpterm` softkeys, position the pointer in an `hpterm` window and press the Menu key. Clicking on a softkey selects that function or setting. Pressing the Menu key again turns off the softkey display.

Additionally, you can color the following elements of `hpterm` softkeys:

- Background.
- Foreground.
- Top shadow.
- Bottom shadow.
- Top shadow tile.
- Bottom shadow tile.

You can automate the coloring process by having the `makeColors` resource automatically select element colors based on a color you specify for the softkey background.

Coloring `hpterm` softkeys is similar to coloring other clients and to coloring the HP Window Manager. You'll find more information about coloring in chapters 5 and 6.

Coloring 'hpterm' Scrollbars

The `hpterm` client also has an option for displaying scrollbars. Scrollbars enable you to scroll the contents of a window, for example, a textfile you are editing. You can specify the color and the width for `hpterm` scrollbars. This is also covered in chapter 5.

Emulating a DEC or Tektronix Terminal

The `xterm` client is a terminal emulation window. `xterm` windows emulate DEC VT102 and Tektronix 4014 terminals. Although `xterm` windows are not the default terminal window for the X Window System, you can use them as your needs require.

Syntax

The syntax of the **xterm** window client is as follows:

xterm [*–options*] [&]

You'll find a list of common viewable client options in "Working with Common Client Options" later in this chapter. For a complete list of **xterm** options, see the **xterm** pages in the reference section.

Using 'xterm' Scroll Features

The **xterm** client has a "jump scroll" option (**-j**). The option enables **xterm**, when its scrolling gets behind, to scroll (jump) several lines at a time from the top of the window.

Another option (**-s**), enables **xterm** to scroll asynchronously. This enables **xterm** to scroll faster when the window screen is no longer up to date due to a high network load.

To use either option, include the option on the command line after the name of the client.

Using 'xterm' Menus

The **xterm** client has three menus. The standard **xterm** menu pops up when the "control" key and the select button are pressed while the pointer is inside the **xterm** window. The "Modes" menu pops up when the "control" key and the alternate button are pressed while the pointer is in the window. The "Tektronix" menu pops up when the "control" key and the alternate button are pressed in a Tektronix window.

Special Terminal Emulator Options

Both **hpterm** and **xterm**, because they are terminal emulators, have some special options that other clients don't have.

Making a Login Window

Both **hpterm** and **xterm** have an option that allows you to specify that the window runs a login shell before displaying the command-line prompt. Using

the `-ls` option, you can have the window display a login prompt, and ask for a login name and password, before the window displays the command prompt.

Cutting and Pasting with the Mouse

Both `hpterm` and `xterm` allow you to use the mouse for cut and paste operations. You can cut text from one location in a window to another, or from one window to another.

Currently, `hpterm` and `xterm` use the button definitions in the following table for cut and paste operations:

Table 4-6. Mouse Button Definitions for Cut and Paste Operations.

If you see ...	On a 3-button mouse press ...	On a 2-button mouse press ...
Button 1	The left button.	The left button.
Button 2	The middle button.	Both buttons simultaneously.
Button 3	The right button.	The right button.

To cut and paste using `hpterm`, use the following procedures:

Cutting text — To cut text, follow these steps:

1. Press *and hold* the Shift key.
2. Position the pointer at the start of the text you want to cut and press *and hold* button 2. This marks the beginning of the text region.
3. Drag the pointer to the end of the text you want to cut and release the button. This copies the text into a global **cut buffer,** a buffer that holds text that has been edited out.

Pasting text — To paste text from the global cut buffer into a window, follow these steps:

1. Press *and hold* the Shift key.
2. Position the pointer in the window where you want to paste the text. The text will appear like it is being

typed at the cursor's location, so you may need to position the cursor as well.

3. Click button 3 to "type" the text.

Copying a line

To copy a line of text from one place and paste it in at the cursor location, follow these steps:

1. Press *and hold* the Shift key.
2. Position the pointer at the start of the text you want to copy.
3. Click the select button to copy text from the pointer to the end of the line.
4. Click button 3 to "type" the text.

To cut and paste using `xterm` use the following procedures:

Cutting text

To cut text, follow these steps:

1. Position the pointer at the start of the text you want to cut.
2. You can cut a text region in the following three ways:
   - To cut a region character by character, click *and hold* button 1.
   - To cut a region word by word, double-click *and hold* button 1.
   - To cut a region line by line, triple-click *and hold* button 1.

   This marks the beginning of the text region.
3. Drag the pointer to the end of the text you want to cut and release the button. This copies the text into the global cut buffer.

Pasting text

To paste text from the global cut buffer into a window, follow these steps:

1. Position the pointer in the window where you want to paste the text. The text will appear like it is being

typed at the cursor's location, so you may need to position the cursor as well.

2. Click button 2 to "type" the text.

Extending text

You can extend or contract either half of the current selection by following these steps:

- To extend or contract the first half of the selected text, follow these steps:

  1. Position the pointer in the first half of the text that you have selected with the select button.
  2. Press *and hold* button 3.
  3. To expand or contract the first half of the selection, drag the pointer away from or toward the center point of the selection.
  4. When the selection includes the correct text, release button 3.

- To extend or contract the second half of the selected text, follow these steps:

  1. Position the pointer in the second half of the text that you have selected with the select button.
  2. Press *and hold* button 3.
  3. To expand or contract the second half of the selection, drag the pointer away from or toward the center point of the selection.
  4. When the selection includes the correct text, release button 3.

Scrollbars

You can start either an `hpterm` or `xterm` window with scrollbars. To do this, include the `-sb` option on the command line when you start the window. For example, to start an `hpterm` window with a scrollbar, type the following line after the command prompt:

```
hpterm -sb & Return
```

Window Titles and Icon Names

By default the title of a terminal emulation window is **Terminal Emulator**. Equally original are the default names that appear on labels of **hpterm** and **xterm** icons. These are, respectively "hpterm" and "xterm." Two options enable you to give your terminal windows and icons more original names if you so desire.

Use the **-title** option to give a title to a terminal emulation window. Titles with two or more word must be enclosed in quotes (*"title1 title2"*).

Use the **-n** option to give a name to the icon of a terminal emulation window. Icon names of two or more word must be enclosed in quotes (*"name1 name2"*). Note also that lengthy names may be truncated on the right to the width of the label.

The following example illustrates the use of these two options:

```
hpterm -n System -title "System Window" & Return
```

This example creates an **hpterm** window, giving it the title "System Window." When the window is iconified, the icon label reads "System."

Telling Times with 'xclock'

The X Window System includes a clock client called **xclock**. You can choose either an analog clock (a clock with hands and a face) or a digital clock (a clock with a text readout showing the day, date, time, and year).

Syntax

The syntax for **xclock** is as follows:

xclock [*–options*] [*&*]

You'll find a list of options that `xclock` shares with other viewable clients in "Working with Common Client Options" later in this chapter. For a complete list of `xclock` options, see the `xclock` pages in the reference section.

Although ampersand (`&`), strictly speaking, is an option, you will rarely, if ever, find it practical to use `xclock` with out it. When run from the command line as a foreground process (without the `&`), `xclock` takes control of the window and *does not* return the command-line prompt, thus making it impossible for you to use the window until you either close the clock or kill its process.

Some 'xclock' Options

The `xclock` client comes with some options that are unique.

Marking the Half Hours

The `-chime` option causes the speaker on your system to sound once on the half hour and twice on the hour.

Selecting the Clock Format

As mentioned, `xclock` has two formats, analog and digital. The analog format is the default. Specifying the `-analog` format (or no format) draws a conventional 12-hour clock face with strokes marking the hours and ticks marking the minutes. Specifying the `-digital` format draws a digital readout containing the day, date, time, and year.

Updating the Time

The `-update` *seconds* option enables you select the time interval between updates to the clock display. The default is an update every 60 seconds.

Examples

The following examples illustrate both clock formats:

```
xclock -digital -update 10 & Return

xclock -analog -chime -update 5 & Return
```

The first example creates a digital clock that updates every 10 seconds. The second example creates an analog clock that chimes every 30 minutes and updates every 5 seconds.

Viewing System Load with 'xload'

The X Window System includes a client called **xload** that displays a histogram of the current system load.

Syntax and Options

The syntax for **xload** is as follows:

xload [*–options*] [*&*]

You'll find a list of options that **xload** shares with other viewable clients in "Working with Common Client Options" later in this chapter. For a complete list of **xload** options, see the **xload** pages in the reference section.

Like **xclock**, the **&** that completes an **xload** command line is, strictly speaking, an option. But you will rarely find it practical to use **xload** without it. When run from the command line as a foreground process (without the **&**), **xload** *does not* return the command-line prompt, thus making it impossible for you to use the window until you either close or kill the **xload** client.

Some 'xload' Options

The **xload** client comes with some options that are unique.

Updating the Load

The **-update** *seconds* option enables you to select the time interval between updates to the load histogram display. The default is an update every 5 seconds.

Scaling the Histogram Graph

The `-scale` *division* option enables you to adjust the scale of the histogram by drawing extra division lines on the graph. By default `xload` measures the average load on the system using a scale of 0 (no load) to 1 (a single division). Using the `-scale` option, however, you can select a division other than 1 against which to measure the load.

Note that if you use the default setting and the system load goes beyond that, extra divisions will be drawn automatically to keep the load in scale.

Example

The following example illustrates an `xload` client started from the command line:

```
xload  -update  15  -scale  2 &  Return
```

This example creates a load histogram that updates every 15 seconds and uses a scale of 2 units.

Working with Common Client Options

The viewable clients have the following options in common:

- Color.
- Display.
- Size and location.
- Fonts.
- Other options.

Color Options

All viewable clients have elements that you can color. If your system uses a monochrome monitor, it is still possible to use the tiling capability of the HP Window Manager to achieve a pleasing 3-D gray-scale color scheme.

The viewable X11 clients, as you might expect, have options for specifying the color of their elements.

Available Client Color Options

The following table lists the colorable elements of X11 clients.

Table 4-7. Color Options for Viewable X11 Clients.

Option Descriptions		X11 Clients			
To change this ...	**Use this option ...**	**hpterm**	**xterm**	**xclock**	**xload**
Foreground color.	`-fg` *color*	√	√	√	√
Background color.	`-bg` *color*	√	√	√	√
Cursor color.	`-cr` *color*	√	√		
Pointer color.	`-ms` *color*	√	√		
Clock hands color.	`-hd` *color*			√	
Hand edge color.	`-hl` *color*			√	

You can specify an element color on the command line in the following two ways:

- By listing the color name after the option.
- By listing the hexadecimal color value after the option.

The file `/usr/lib/X11/rgb.txt` lists all colors that have "names." Specifying a name after a color option causes the element referred to by the option to display in that color.

For example, the following command line creates an `hpterm` window with a black background and a white foreground:

```
hpterm -bg Black -fg White & Return
```

Using Hexadecimal Color Values on the Command Line

While using color names is an easy way to select colors, you are limited by the number of available names. Fortunately, the use of hexadecimal color values offers a solution. You can specify *any* color, whether it has a name or not, by using a hexadecimal color value. This value corresponds to a particular combination of the primary colors: red, green, and blue.

If you use the C shell (`csh`) a color value consists of a hash mark (#) followed by 1, 2, 3, or 4 hexadecimal digits. If you use the Bourne shell (`sh`) or Korn shell (`ksh`), you must place a backslash (\) before the `#`, so a color value consists of `\#` followed by 1, 2, 3, or 4 hexadecimal digits. You must have the same number of digits *for each* of the primary colors. Thus, valid color values consist of 3, 6, 9, or 12 hexadecimal digits.

For example, #3a3 and #300a00300 are both valid color values for the same color, a shade of green. #000, #000000, #000000000, and #000000000000 all specify the color black. And #fff, #ffffff, #fffffffff, and #ffffffffffff all specify white. The number of digits you use in color values depends on your need for subtle shades of color and the capability of your display hardware.

Examples

As an example of specifying color on a command line, suppose you wanted an analog clock with a plum background, white foreground, and black hands with white edges. You could specify the clock in either of the two following ways:

```
xclock -bg violet -fg white -hd black -hl white & Return
```

or

```
xclock -bg #c5489b -fg #fff -hd #000 -hl #fff & Return
```

For the purposes of this example, plum, white, and black were chosen because they are colors with valid color names in `/usr/lib/X11/rgb.txt`. However, at any time you can specify a unique color (one with no name equivalent), for example a slightly darker plum for the background, by creating your own hexadecimal value as follows:

```
xclock -bg #ba408b -fg white -hd black -hl white & Return
```

Specifying Size and Location on the Command Line

Each client you add to your environment is located at a certain position on the root window. The default position is the upper left corner, but you can place a client anywhere on the root window using the `-geometry` option.

The Syntax of the '-geometry' Option

The `-geometry` option has the following syntax:

`-geometry` *Width*x*Height*[±*column*±*row*]

Width	The width of the window in characters (for terminal windows) or pixels (for other clients). Note that the width of the terminal window that displays varies depending on the font size.
Height	The height of the window in lines (for terminal windows) or pixels (for other clients). The height of a terminal window is also dependent on the size of the font chosen.
column	The column location of the window given in pixels. Plus (+) values refer to the left side of the window. Minus (−) values refer to the right side of the window.
row	The row location of the window given in pixels. Plus (+) values refer to the top of the window. Minus (−) values refer to the bottom of the window.

You have the following choices for defining client size and location:

- Include both the size and location in the command. The window appears as specified.
- Include only the size in the command. The window appears in the specified size at the default location.
- Include only the location in the command. The window appears at the specified location in its default size.
- Include neither size nor location in the command. The window appears in the default size at the default location.

Placing Clients on the Root Window

The following table lists some typical locations for a 1280×1024 high-resolution display.

Table 4-8. Example Locations for an 80×24 X11 Terminal Window.

To position a window here ...	Use this location ...
The upper left corner of the root window.	+1+1
The lower left corner of the root window.	+1−1
The upper right corner of the root window.	−1+1
The lower right corner of the root window.	−1−1
The left side at mid-window.	+1+512
The right side at mid-window.	−1+512
The top of the root window and right of center.	+635+1
Centered at left.	+1+330
Centered at right.	−1+330
Centered in the root window.	+320+330

Note

The resolution of screens vary. Some locations may work for you but be off the screen for someone else! Therefore, you may need to experiment, altering the geometry specifications to fit the resolution of the screen.

Example

The following examples illustrate a typical command-line use of the geometry option:

```
xclock -geometry 90x90-1-30 & Return

xload -geometry 120x90+1-1 & Return
```

The first example starts an **xclock** client. The geometry option gives the clock a 90-pixel by 90-pixel size and locates it 1 pixel to the left and 30 pixels up from the lower right corner of the screen.

The second example starts an **xload** client. The geometry option gives the client a 120-pixel by 90-pixel size and locates it in the lower left corner of the screen.

Specifying the Display on the Command Line

As described above in "Starting Remote Clients," you can start an X client program on one computer and have the output of the program display on another. The default display is obtained from the **DISPLAY** environment variable of the system on which the client starts, but the **DISPLAY** variable can be reset dynamically for a client by including a **-display** option on the command line when you start the client.

The Syntax for the '-display' Option

The **-display** option has the following syntax:

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

host	Specifies the hostname of a valid system on the network. Depending on the situation, this could be your system's hostname, or the hostname of a remote system.
display	Specifies the number of the display on the system on which you want the output to appear. On HP 9000 S300's, this number will usually be 0. On HP 9000 S800's, this number could be 0, 1, 2, or 3 depending on the configuration.
screen	The number of the physical CRT screen where the output is to appear. This number is 0 for default one-headed configurations.

Example

An example of using the display option on the command line is the following:

```
hpterm  -display  hpcvfaa:0.0 & Return
```

This command, when issued at a command-line prompt, starts an **hpterm** window on the local system and displays output (the window) on screen 0, display 0 of the **hpcvfaa** system. The window has the default size, location, and color.

Specifying the Font in the Command Line

In addition to the options discussed above, the viewable clients also have an option that enables you to specify the font for text. The **-fn** option enables you to select a font for the hostname that displays on the **xload** client as well as the text for terminal emulation windows.

Selecting a Font

The following table lists X Window System fonts. For best results, type the font name exactly as it appears below.

Table 4-9. X11 Fonts*.

Fonts						
6x10	6x12	6x13	8x13	8x13bold	9x15	9x16apl
9x16bas	9x21apl	9x21bas	9x21ibm	12x21apl	12x21bas	12x28apl
12x28bas	12x28ibm	a14	apl-s25	calc.12x16	calc.6x8	chp-s25
chs-s50	cr.12x20	cr.12x20b	cursor	cyr-s25	cyr-s30	cyr-s38
fcor-20	fg-13	fg-16	fg-18	fg-20	fg-22	fg-25
fg-30	fg-40	fg1-25	fgb-13	fgb-25	fgb1-25	fgb1-30
fgi-20	fgi1-25	fgs-22	fixed	fqxb-25	fr-25	fr-33
fr1-25	fr2-25	fr3-25	frb-32	fri-33	fri1-25	ger-s35
grk-s25	grk-s30	hbr-s25	hbr-s40	hp8.10x20	hp8.10x20b	hp8.12x15
hp8.6x13	hp8.6x13b	p8.6x8	hp8.6x8b	hp8.7x10	hp8.8x16	hp8.8x16b
hp8.8x16i	ipa-s25	iso1.13	iso1.13b	iso1.15	iso1.16	iso1.16b
iso1.20	iso1.20b	iso1.8	k14	kana.10x18	kana.10x20	kana.12x24
kana.8x16	kana.8x18	kana14	krivo	lat-s30	line.8x16	math.18x30
math.6x8	math.8x16	met25	micro	oldera	pica.18x30	plunk
r14	rot-s16	sans12	sansb12	sansi12	serif10	serif12
serifb10	serifb12	serifi10	serifi12	sub	subsub	sup
supsup	swd-s30	sym-s25	sym-s53	variable	vbee-36	vctl-25
vg-13	vg-20	vg-25	vg-31	vg-40	vgb-25	vgb-31
vgbc-25	vgh-25	vgi-20	vgi-25	vgi-31	vgl-40	vgvb-31
vmic-25	vr-20	vr-25	vr-27	vr-30	vr-31	vr-40
vrb-25	vrb-30	vrb-31	vrb-35	vrb-37	vri-25	vri-30
vri-31	vri-40	vsg-114	vsgn-57	vshd-40	vtbold	vtsingle
vxms-37	vxms-43	xif-s25				
*Font names are shown without extensions.						

Working with Fonts

Fonts are kept in the `/usr/lib/X11/fonts` directory. Although it is always a good idea to give a complete path to the font you want to use, if you don't, the server looks in the `/usr/lib/X11/fonts` directory by default. If it doesn't find the specified font there, it tries (in most cases) to substitute the `fixed` font.

When specifying a font, you only need to specify the font name, not the extension.

The two terminal emulators also have a `-fb` option. You can use this option to specify a bold text. The text specified must be the same height and width as the font specified with `-fn`, the "normal" font.

Example

The following examples illustrate the command-line use of the font option:

```
hpterm  -fn  iso1.20 &  Return

hpterm  -fn  isp1.20  &  Return
```

The first line creates an `hpterm` window with a large, easy-to-read font (iso1.20). The second line represents a misspelling of the first line. The result is the creation of a window, but the font used for the command-line prompt is the default font, not iso1.20.

Where to Go Next

If your X Window System environment meets your present needs, you can stop here. If, however, you would like to customize your environment a little, perhaps coordinate the colors of your clients, or select different clients to display when you start X, or arrange them more efficiently on your root window, you should continue to chapter 5.

Chapter 6 explains, in more detail than the average mortal need be concerned about, how to work with the HP Window Manager and its resources to fine-tune your control over your X environment. Chapters 7, 8, and 9 present cases where customization is needed because of special hardware considerations or the extensive use of graphics.

5

Customizing Your Local X Environment

As you become familiar with the X Window System, you will probably want to modify your X environment to better suit your situation. Chapter 5 discusses customizing your window environment. Using the information in this chapter, you can change the appearance and behavior of the X Window System to suit your needs *without affecting the needs of other users.* These changes include the following:

- Customizing the colors of clients.
- Changing the clients that start when you start X.
- Modifying HP Window Manager menus.
- Starting X at login.
- Creating custom bitmaps.
- Customizing the root window.
- Working with fonts.
- Using Remote Hosts.

Before You Begin Customizing

To customize your window environment, you must modify or create three configuration files. These files contain information that the X server uses to configure your window environment. Incorrectly modifying these files could bring your X Window System to a screeching halt. So if you are new to this type of thing, read the following two sections. They list some simple safety precautions (often overlooked by people who "know what they're doing") that keep you from getting into trouble if you make a mistake. They also give you a little background on the configuration files with which you'll be working.

How to Begin Customizing

Swimming pools you should jump into with both feet; customizing your environment you should approach step by step. Although the following safety tips may take a little more time to implement, they are the steps that people regretfully "wish they had taken" after something has gone wrong.

Making Backup Copies of Your Work

Don't modify any original files. Make a copy of the original file and then modify the copy. That way, if all else fails (and it sometimes does), you can go back and get another copy of the original and start again. As you get deeper into rearranging your environment, test your modifications and, if they work properly, save that version of your modifications, make a copy of it, and continue the rest of your modifications on the copy.

Making Incremental Changes

Make incremental changes when you edit the configuration files. That way, if something goes wrong, you can easily isolate where the mistake is. It's much easier to pinpoint a mistake in syntax or spelling if you've only modified one line of one file, rather than multiple lines in several files.

Choosing a Text Editor

The three configuration files are ASCII text files. You can use `vi` or `emacs` or any other editor that produces ASCII text files to do your editing. You edit

the text of the configuration files just like you would edit the text of a letter, replacing what you don't want with something more appropriate.

One trick that you might consider is to comment out a line that you don't want rather than deleting it from the file. To comment out a line, place a hash mark or pound sign (#) in the left margin of the line (use a ! to comment out a line in `.Xdefaults` or `sys.Xdefaults`). This allows you to use the line as a model for future editing and provides you with the opportunity to restore it (by uncommenting it) at some future time.

Where to Begin Customizing

Three configuration files come with the X Window System:

- sys.Xdefaults
- sys.x11start
- system.hpwmrc

You'll find these files in the `/usr/lib/X11` directory. The files supply system-wide default configuration for users who start X but don't have individual configuration files in their home directories.

The following three configuration files should be in your home directory if you want to customize your X environment. Typically, you copy them from their system-wide versions in `/usr/lib/X11`:

`.Xdefaults`	Specifies default appearance and behavior characteristics for clients.
`.x11start`	Specifies the clients that start when the X Window System starts.
`.hpwmrc`	Specifies the menus, menu selections, and button and keyboard bindings that control the HP Window Manager.

Note that the `/usr/lib/X11/app-defaults/` directory may also contain configuration files for client applications.

Customizing the Colors of Clients

You control the color of the clients (including the HP Window Manager) that display in your X environment by modifying the `.Xdefaults` file. Valid color names are stored in `/usr/lib/X11/rgb.txt`.

Copying 'sys.Xdefaults' to '.Xdefaults'

When you issue the `x11start` command to start the X Window System, the command looks in your home directory for a `.Xdefaults` file. If it finds the file, it uses the information in the file to color your X environment. If it doesn't find the file, the `x11start` command uses `/usr/lib/X11/sys.Xdefaults`.

To begin customizing the colors of your X environment, copy the `sys.Xdefaults` file to your home directory as `.Xdefaults`. The following copy command assumes that you are in your home directory when you issue it:

```
cp  /usr/lib/X11/sys.Xdefaults  .Xdefaults  Return
```

This gives you a read-only copy of `.Xdefaults`. You must make the `.Xdefaults` file writable so that you can modify it. To do this, type the following command:

```
chmod  u+w  .Xdefaults  Return
```

This will enable you to color the clients in your environment without affecting the environments of other users on the system.

If, during the editing process, the file becomes corrupted and inoperable, you can always make a fresh copy from `/usr/lib/X11/sys.Xdefaults` and begin the editing process again.

Changing Client Colors

Changing the color of a particular client element is a simple process. You specify a value for the resource that controls the element you want to color. Use the following steps:

1. Start your text editor and open the `.Xdefaults` file.
2. Scroll down or search for the *client*resource* you want to color.

3. Delete the ! and the space from the left margin to activate the line.
4. Replace the "<color>" at the end of the line with the color you desire.
5. Save the file and exit the text editor.

To view the effect of a change to `.Xdefaults`, simply start a client of the type whose color you modified.

Determining Which Elements to Color

The following tables list the colorable elements of your X environment by client.

Table 5-1. Terminal Window Elements.

To color this element ...	Look for this resource ...
hpterm window text	! HPterm*foreground:
hpterm window background	! HPterm*background:
hpterm window text cursor	! HPterm*cursorColor:
hpterm window mouse pointer	! HPterm*pointerColor:
xterm window text	! Xterm*foreground:
xterm window background	! Xterm*background:
xterm window text cursor	! Xterm*cursorColor:
xterm window mouse pointer	! Xterm*pointerColor:

Table 5-2. Load Histogram Elements.

To color this element ...	Look for this resource ...
system load histogram foreground	! Xload*foreground:
system load histogram background	! Xload*background:

Table 5-3. Clock Elements.

To color this element ...	Look for this resource ...
analog clock tick marks	! XClock*foreground:
digital clock text	! XClock*foreground:
clock background	! XClock*background:
clock hands	! XClock*hands:
edges of clock hands	! XClock*highlight:

Table 5-4. Window Frame Elements.

To color this element ...	Look for this resource ...
window frame background	! Hpwm*background:
window frame text	! Hpwm*foreground:
top and left window frame bevel	! Hpwm*topShadowColor:
bottom and right window frame bevel	! Hpwm*bottomShadowColor:
active window frame background	! Hpwm*activeBackground:
active window frame text	! Hpwm*activeForeground:
top and left active window beveling	! Hpwm*activeTopShadowColor:
bottom and right active window beveling	! Hpwm*activeBottomShadowColor:

Syntax

At some point, you may want to change the color of an element that is not in your `.Xdefaults` file. You can add that element to the file by typing it in `.Xdefaults` on a line by itself using the following syntax:

$$client*resource: \left\{ \begin{array}{l} colorname \\ \#hexadecimal \end{array} \right\}$$

For *client*, you can use any valid viewable X client. For *resource*, you can use any valid color resource for that client. The surrounding lines in the file

provide you with examples to model your line after. You can find a complete list of the resources for each client in the reference section.

The color you specify can be either a colorname from the **/usr/lib/X11/rgb.txt** file or a hexadecimal value. While colornames are easier to remember, hexadecimal values enable you to specify a greater variety of colors.

A hexadecimal value is composed of three segments, one segment for each of the primary colors red, green, and blue. A hexadecimal value consists of a hash mark (**#**), signaling the start of a hexadecimal number, followed by 1, 2, 3, or 4 hexadecimal digits *for each* primary color. Thus a valid color value can be 3, 6, 9, or 12 hexadecimal digits.

Examples

The following examples illustrate some typical lines in your **.Xdefaults** that color client elements.

```
XClock*foreground:    Black
XClock*background:    White
XClock*hands:         SkyBlue
XClock*highlight:     Black
```

The above lines color the elements of **xclock** clients. The first line makes the tick marks of analog clocks (and the readout of digital clocks) black. The next line gives them white backgrounds (faces). The next two lines color the hands of analog clocks skyblue with black borders.

When coloring client elements, you should usually color adjacent elements in contrasting colors. The obvious mistake is coloring clock hands the same color as the background. Sure, the hands display in the color you select, but it's frightfully hard to tell the time. The same holds true for foregrounds and backgrounds that lack sufficient contrast.

What Colors Are Available

You can color your X11 environment by specifying any of the color names listed in the following table. Type the color name exactly as it appears below.

Table 5-5. X Window System Color Name Table.

Available Colors			
Aquamarine	Black	Blue	BlueViolet
Brown	CadetBlue	Coral	CornflowerBlue
Cyan	DarkGreen	DarkOliveGreen	DarkOrchid
DarkSlateBlue	DarkSlateGray	DarkSlateGrey	DarkTurquoise
DimGray	DimGrey	Firebrick	ForestGreen
Gold	Goldenrod	Gray	Green
GreenYellow	Grey	IndianRed	Khaki
LightBlue	LightGray	LightGrey	LightSteelBlue
LimeGreen	Magenta	Maroon	MediumAquamarine
MediumBlue	MediumForestGreen	MediumGoldenrod	MediumOrchid
MediumSeaGreen	MediumSlateBlue	MediumTurquoise	MediumVioletRed
MidnightBlue	Navy	NavyBlue	Orange
OrangeRed	Orchid	PaleGreen	Pink
Plum	Red	Salmon	SeaGreen
Sienna	SkyBlue	SlateBlue	SpringGreen
SteelBlue	Tan	Thistle	Transparent
Turquoise	Violet	VioletRed	Wheat
White	Yellow	YellowGreen	

Where to Find the Available Color Names

All of the color names available in the X Window System are listed in the `/usr/lib/X11/rgb.txt` file. You can find the names of colors by typing the following command to view the file:

```
more /usr/lib/X11/rgb.txt Return
```

The file is several "pages" long, so you may find it more convenient to make a printed copy of the file using the following command:

```
pr -l60 -h "X11 Color Table" /usr/lib/X11/rgb.txt | lp Return
```

Coloring the HP Window Manager Automatically

The HP Window Manager contains three resources that you can use to automatically color the elements of the window frame to achieve a pleasing 3-D effect. These resources have the following syntax:

```
Hpwm*makeColors:       { all       }
                       { shadow    }
                       { one       }

Hpwm*makeActiveColors: { all       }
                       { shadow    }
                       { one       }

Hpwm*background:       { color     }
                       { #hexadecimal }
```

The `makeColors` resource uses the color specified with `Hpwm*background` to generate colors for the frame elements of inactive windows. The `makeActiveColors` resource uses the `Hpwm*background` color to generate colors for the frame elements of the active window.

The following table lists the values for the two resources:

Table 5-6. The Values for Automatically Coloring Frame Elements.

To color these elements ...	Use this value ...
Top and bottom shadow, and foreground (tile elements for top and bottom shadows are set to foreground).	`all`
Just top and bottom shadow.	`shadow`
No elements automatically colored. You must specify individual colors or accept the default color for an element.	`none`

Determining Where to Color Your Environment

The usual place to specify colors is in the `.Xdefaults` file in your home directory. However, you can change the color of a particular instance of an element (such as the foreground color of a single window) by specifying that color on the command line that starts the client. If you start the client when you start X11, the command line would be in the `.x11start` file. If you start the client from a menu, the command line would be in the `.hpwmrc` file.

For example, if you wanted an `hpterm` window to have a DarkSlateGrey background and White foreground, you could specify these colors on the command line you used to start the window.

Coloring a Single Instance of a Client

The following command, issued at the command line prompt, overrides any background and foreground colors specified in the `.Xdefaults` file and creates a single `hpterm` window with a DarkSlateGray background and White foreground.

```
hpterm -bg DarkSlateGrey -fg White & Return
```

This syntax should be familiar to you if you read chapter 4.

Coloring Windows that Start Automatically

The following line in your `.x11start` file overrides any background and foreground colors specified in the `.Xdefaults` file and creates an `hpterm`

window with a DarkSlateGrey background and White foreground *each time* you start X11.

```
hpterm -bg DarkSlateGrey -fg White &
```

Note that the syntax of the above example is exactly like the syntax used when you start a client from the command line.

Coloring Windows that Start from Menus

The following line in your `.hpwmrc` file overrides any background and foreground colors specified in the `.Xdefaults` file and, when you choose the **Dark Window** selection from the menu, creates an **hpterm** window with a DarkSlateGrey background and White foreground.

```
"Dark Window" f.exec "hpterm -bg DarkSlateGrey -fg White &"
```

This syntax is similar to the command-line syntax with which you are already familiar. You'll learn more about it in "Modifying HP Window Manager Menus" later in this chapter.

Coloring 'hpterm' Softkeys and Scrollbars

To color **hpterm** softkeys or scrollbars, you may need to add one or more lines from the following table to your `.Xdefaults` file:

Table 5-7. You Can Color These 'hpterm' Softkey and Scrollbar Elements.

To color this element ...	Add this line ...
softkey text	HPterm*softkey*foreground:
softkey background	HPterm*softkey*background:
top and left softkey bevel	HPterm*softkey*topShadowColor:
bottom and right softkey bevel	HPterm*softkey*bottomShadowColor:
top and left softkey bevel tile	HPterm*softkey*topShadowTile:
bottom and right softkey bevel tile	HPterm*softkey*bottomShadowTile:
scrollbar background	HPterm*scrollBar*foreground:
scrollbar background	HPterm*scrollBar*background:

The lines you add to the `.Xdefaults` file all have the following syntax:

$$\texttt{HPterm*}\left[\begin{matrix}\texttt{softkey}\\ \texttt{scrollBar}\end{matrix}\right]\texttt{*}\textit{resource}\texttt{:}\ \left\{\begin{matrix}\textit{color}\\ \textit{\#hexadecimal}\end{matrix}\right\}$$

The color you select can be either a color name (Magenta) from the `rgb.txt` file or a hexadecimal value (#ffe000ffe).

For tile, you can select a number of tile "patterns." For a complete list see "Changing the Tile of Window Frames" in Chapter 6.

Changing the Clients that Start When You Start X

By modifying the `.x11start` file in your home directory, you can control which clients display as part of your environment when you start X.

Copying 'sys.x11start' to '.x11start'

The clients that start by default when you start X are specified by command lines in the `sys.x11start` file. To change the clients that start in your personal X environment from the default (`hpwm` and an `hpterm` window), copy `sys.x11start` from the `/usr/lib/X11` directory to your home directory. The following command assumes you are in your home directory:

```
cp  /usr/lib/X11/sys.x11start  .x11start  Return
```

This gives you a read-only copy of `.x11start`. You must make the `.x11start` file writable so that you can modify it. To do this type, the following command:

```
chmod  u+w  .x11start  Return
```

This will enable you to change the clients that start in your environment without affecting the environments of other users on the system.

If, during the editing process, the `.x11start` file becomes corrupted and inoperable, you can always make a fresh copy from `/usr/lib/X11/sys.x11start` and begin the editing process again.

Viewing X11 Start Error Messages

The **x11start** command records any messages that occur as X11 starts. Viewing these messages is an important tool for finding errors in your configuration files. The start command puts messages in the **.x11startlog** file in your home directory.

If you start X11 and your environment displays as expected, no error messages will be generated and **.x11startlog** will be empty.

However, at some point you may start X11 and your environment does *not* display as expected. For example, maybe one of your terminal windows doesn't display. To view any error messages that occurred, type the following at the command-line prompt in your home directory:

```
more .x11startlog Return
```

Any error messages in the file will be listed on the screen and, although decidedly cryptic in nature, they at least provide a starting place for locating the cause of the error.

Starting a Different Window Manager

The HP Window Manager is the default window manager of your X Window System. However, another window manager, **uwm**, is included with the X Window System.

The **uwm** manager features frameless windows and comes with a single menu. For more information about **uwm** see chapter 6, "Managing Windows with 'uwm'." If you like the HP Window Manager, but would rather forgo the window frame, see chapter 6, "Using Windows without Frames."

To use the **uwm** window manager instead of the HP Window Manager, follow these steps:

1. Start your text editor and open the **.x11start** file.
2. Scroll down or search for the line that reads as follows:

   ```
   hpwm  $@ & # Start the HP Window Manager
   ```

3. Comment out this line by typing a **#** and a space in the left margin. Optionally, you can delete the entire line (not recommended).

4. On a new line at the same location (before the line that begins with `sleep 5 ...`), type the following command:

```
uwm  $@  & #Start uwm window manager
```

5. Save the file and exit the editor.

To put the `uwm` window manager into effect, exit the X Window System by pressing CTRL Left Shift Reset. Then restart the window system again.

Starting Programs Automatically

If you'd like to start more than `hpwm` and a single `hpterm` window when you start X11, you need to add a few more lines to your `.x11start` file. One line for each client or non-client you want to start.

Syntax and Examples

The syntax for starting a program automatically matches the syntax for running the program from the command-line prompt:

client [*–options*] [&]

Starting Clients

Follow these steps to add other clients to your X11 environment:

1. Start your text editor and open `.x11start`.
2. Scroll down or search for the line that reads as follows:

```
hpterm -C -geometry 80x24+1+1 $@ &
```

3. On the lines below this, insert command lines for each client you want to start, one client per line.
4. When you're done, check your syntax and spelling. If all is correct, save the file and exit the editor.

For example, the following two lines start a clock and an `hpterm` window as part of the initial X environment:

```
xclock -digital -update 10 -geometry 160x30-1+1 &
hpterm -geometry 80x24-1-1 &
```

The first line adds a 160×30 pixel digital clock to the upper right corner of the screen. The clock is updated every 10 seconds. The second line starts an 80 column by 24 line `hpterm` emulation window in the lower right corner of the screen. Both clock window and `hpterm` window are the default colors specified in `.Xdefaults` or `/usr/lib/X11/sys.Xdefaults`.

Note that both lines end with an ampersand (`&`), telling the system to start these clients as background processes. Note also that the geometry dimensions of clients like the clock are in *pixels;* however, the dimensions of terminal windows are in columns (characters across) and lines (characters down).

Starting Non-Clients

Starting non-clients (commands or programs) automatically is similar to starting clients. Follow these steps to add non-clients to your X11 environment:

1. Start your text editor and open `.x11start`.
2. Scroll down or search for the line that reads as follows:

   ```
   hpterm -C -geometry 80x24+1+1 $@ &
   ```

3. On the lines below this, insert command lines for each non-client you want to start, one non-client per line. Remember that a non-client, because it does not create its own window, is started by the `-e` option (`e` for "execute") from an `hpterm` or `xterm` window.
4. When you're done, check your syntax and spelling. If all is correct, save the file and exit the editor.

For example, the following two lines start `mailx`, an electronic mail program, and login to a remote host, `hpcvfaa`:

```
hpterm  -e  mailx  &
hpterm  -e  rlogin  hpcvfaa  &
```

Both windows that contain the two non-clients are the default size and colors. Notice also that, in this example, they are *both* at the default location, so first one appears and then the other appears right over it – usually not the best practice. A better way is to include a geometry option for one or both windows. Another alternative is to use a `-iconic` option for one window:

```
hpterm  -iconic  -e  mailx  &
hpterm  -e  rlogin  hpcvfaa  &
```

This modified example starts the `mailx` window as an icon. Only when you want to read mail do you need to change the icon into a window.

Discovering Your Options

The following tables repeat the client option information from chapter 4 so you can avoid excessive page churning caused by flipping back and forth.

Table 5-8. Color Options for Viewable X11 Clients.

Option Descriptions		X11 Clients			
To change this ...	**Use this option ...**	**hpterm**	**xterm**	**xclock**	**xload**
Foreground color.	`-fg` *color*	√	√	√	√
Background color.	`-bg` *color*	√	√	√	√
Cursor color.	`-cr` *color*	√	√		
Pointer color.	`-ms` *color*	√	√	√	√
Clock hands color.	`-hd` *color*			√	
Hand edge color.	`-hl` *color*			√	

Table 5-9. Other Options for Viewable X11 Clients.

Option Descriptions		X11 Clients			
To change this ...	**Use this option ...**	**hpterm**	**xterm**	**xclock**	**xload**
Client location.	`-geometry` *w×h±col±row*	√	√	√	√
Font Displayed.	`-fn` *font*	√	√	√	√
Update interval.	`-update` *number*			√	√
Clock chime.	`-chime`			√	
Analog clock.	`-analog`			√	
Start a program.	`-e` *command*	√	√		
Name of icon.	`-n` *name*	√	√		
Title of window.	`-title` *title*	√	√		
Window name.	`-name` *name*	√	√		
Start client as icon.	`-iconic`	√	√		
Where client displays.	`-display` *host:display.screen*	√	√	√	√

You can control the size and location of each viewable client you add to your `.x11start` file using the `-geometry` option. If you don't specify a `-geometry` option, the client appears in the default size and at the default location.

The syntax of the `-geometry` option is as follows:

`-geometry` *width×height*[*±column±row*]

The size, *width×height*, is in characters by lines for terminal windows, and in pixels for clocks and load histograms. The location, *±column±row*, is in pixels and depends on the resolution of your screen. Plus values (+) start at the upper left corner of the screen and proceed down and to the right. Minus values (−) start at the lower right corner of the screen and proceed up and to the left.

The following table lists some typical locations for a 1280×1024 high-resolution display.

Table 5-10. Sample Locations for an 80×24 X11 Terminal Window.

To position a window here ...	Use this location ...
The upper left corner of the root window.	+1+1
The lower left corner of the root window.	+1−1
The upper right corner of the root window.	−1+1
The lower right corner of the root window.	−1−1
The left side at mid-window.	+1+512
The right side at mid-window.	−1+512
The top of the root window and right of center.	+635+1
Centered at left.	+1+330
Centered at right.	−1+330
Centered in the root window.	+320+330

The options listed here are some of the more commonly used ones. For a complete list of options for each client, see that client's pages in the reference section.

Modifying HP Window Manager Menus

The HP Window Manager menus are controlled by an ASCII text file in the `/usr/lib/X11` directory called `system.hpwmrc`, unless you have a file in your home directory called `.hpwmrc`. You can add or delete menu selections by copying `system.hpwmrc` to your home directory as `.hpwmrc` and modifying it to suit your needs.

Copying 'system.hpwmrc' to '.hpwmrc'

The following command line copies the `system.hpwmrc` file to your home directory as `.hpwmrc`. The command assumes that you are in your home directory when you issue it.

```
cp /usr/lib/X11/system.hpwmrc .hpwmrc Return
```

This gives you a read-only copy of `.hpwmrc`. You must make the `.hpwmrc` file writable so that you can modify it. To do this, type the following command:

```
chmod u+w .hpwmrc Return
```

Having a `.hpwmrc` file in your home directory enables you to change the appearance and behavior of your window manager without affecting the environments of other users on the system.

Syntax

All HP Window Manager menus have the following syntax:

```
Menu MenuName
{

    selection1    function   ["arguments"]
    selection2    function   ["arguments"]
    selection3    function   ["arguments"]
    selection*    function   ["arguments"]
}
```

Each line identifies a selection name followed by the function to be done if that selection is chosen and any arguments to the function. Separate selections, functions, and arguments with either a space or a tab. The order of the selections is the order of their appearance when you display the menu.

Adding Selections

The HP Window Manager comes with two menus, a system menu for each window and a root menu for screen operations. You can modify either menu. However, for consistent operation between systems, it is usually better to modify the root menu.

To add menu selections to a menu, use the following steps:

1. Start your text editor and open the file `.hpwmrc`.
2. Scroll down or search for the line that reads as follows (for the root menu):

   ```
   "Root Menu" f.title
   ```

   This is the title of the root menu. Below it, in order of appearance, are the menu selections.
3. On a line below this, depending on the selection order you want, insert the line you want to add.
4. Check your modification and, if it is correct, save the file and exit the editor.

Deleting Selections

Follow these steps if you want to delete a selection from a menu:

1. Start your text editor and open the file `.hpwmrc`.
2. Scroll down or search for the line you want to delete.
3. Delete the line (you can also comment out the line using a `#`).
4. Save the file and exit the text editor.

Examples

The following example modifies the default root menu:

```
Menu DefaultRootMenu
{
 "Root Menu"        f.title
 "Create Window"    f.exec "hpterm &"
 "Read/Send Mail"   f.exec "hpterm -geometry 80x24+320+330 -e mailx &"
 "Goto hpcvfaa"     f.exec "hpterm -e rlogin hpcvfaa &"
 "Refresh Root"     f.refresh
 "Shuffle Up"       f.circle_up
 "Shuffle Down"     f.circle_down
 "Refresh"          f.refresh
 no-label           f.separator
 "Restart"          f.restart
}
```

The modifications to this root menu remove the clock and load histogram, and add selections for electronic mail and a remote login to **hpcvfaa**.

Note that arguments to the **f.exec** function are enclosed in quotes (" "). *This is a must* because of the number of arguments that follow the function. Similarly, enclose in quotes any selection name that is two or more one words.

Viewing the Results of Your Modification

To view the results of your modification, display the root menu and drag the pointer down the menu until you highlight the "Restart" selection. Then release the button.

When the HP Window Manager is completely restarted, you can display the root menu, and your changes will be apparent.

Starting X11 at Login

You can configure your system to start the X Window System at login in two ways:

- Use the HP-UX `reconfig` program to add a new account to the system, specifying "X11 Windows" at login.
- Edit your existing `.login` or `.profile` login file to include the `x11start` command.

If your login isn't already configured to start X11 automatically, you can edit your login file to do so. If you are adding a new login to your system, you can use the `reconfig` program to tell the system you want X11 at login.

The rest of this section explains how, if you currently have a login on the system, you can edit your `.login` or `.profile` file so that when you log in, your X environment starts automatically.

Modifying Login Files

Which login file you edit depends on which shell command interpreter you use. If you use the C shell, edit `.login`. If you use the Bourne or Korn shell, edit `.profile`.

Finding Out Which Shell You Use

If you are not familiar with which shell you use, type:

env **Return**

This command lists your environment variables. Look for the one named `SHELL`.

If you see ...	Edit ...
`SHELL=/bin/csh`	.login
`SHELL=/bin/sh`	.profile
`SHELL=/bin/ksh`	.profile

Editing the File

Once you have determined the proper login file to edit, use **vi** or some other editor that produces ASCII text files to make the following modification to the bottom of the file.

If you use the C shell, follow these steps to modify your **.login** file:

1. Copy your original **.login** to **.login.old**(just in case).
2. Start your text editor and open **.login**.
3. Page or scroll down to the bottom of the file.
4. Insert the following lines at the bottom of the file:

```
if ("`who am i | grep console`" != "" ] then
    exec /usr/bin/x11start
endif
```

5. Save your edited file and exit the text editor.

If you use either the Bourne or the Korn shell, follow these steps to modify your **.profile** file:

1. Copy your original **.profile** to **.profile.old** (just in case).
2. Start your text editor and open **.profile**.
3. Page or scroll down to the bottom of the file.
4. Insert the following lines at the bottom of the file:

```
if [ "`who am i | grep console`" != "" ]
then
    exec /usr/bin/x11start
```

5. Save your edited file and exit the text editor.

These lines verify that you are logging in from the console, and not from a remote location, before starting X11 on your system. This avoids the possibility of undesirable effects caused by inadvertently starting X on your system from a remote login.

Viewing the Result of Your Edit

To view the result of your edit, exit the X Window System by pressing CTRL Left Shift Reset simultaneously. Remember to use the left Shift key.

When the command-line prompt returns to the screen, you can either log out and then log back in, or type `source .login` (if you use the C shell), `source .profile` (if you use the Bourne shell), or `. .profile` (if you use the Korn shell) to restart X11.

After a few seconds, your system should start the X Window System. From then on, whenever you login, X11 will start automatically.

Using the 'reconfig' Program

You can also start the X Window System automatically for a new user by adding the user with the `reconfig` program.

Before you run `reconfig`, you must be superuser. As one of the options of "Add a new user", you can select either the X Window System or HP Windows 9000 to start at login. Select the X Window System.

Creating Custom Bitmaps with 'bitmap'

Using the `bitmap` client, you can create your own custom bitmaps and use them to tile your root window, for custom root-window cursor shapes, or for custom menu selections.

Syntax and Options

The syntax for `bitmap` is as follows:

```
                ┌-help                                  ┐
                |-display host:display.screen           |
                |-geometry w×h±col±row                  |
                |-nodashed                              |
bitmap          |-fn font                               |  filename  Width×Height
                |-fg color                              |
                |-bg color                              |
                |-hl color                              |
                └-ms color                              ┘
```

-help	Prints a summary of the command usage.
-display	Specifies the screen where **bitmap** is to appear.
-geometry	Sets the size of the **bitmap** window to the specified width×height and locates the window at the specified column and row.
-nodashed	Specifies the **bitmap** grid should not be made of dashed lines.
-fn	Specifies the font to use in the **bitmap** command panel labels.
-fg	Specifies the foreground color.
-bg	Specifies the background color.
-hl	Specifies the color of the highlight used to mark the center of a circle, the hot spot, and move areas.
-ms	Specifies the color of the pointer.
filename	Specifies the name of the bitmap file to open or create.
Width×Height	The size of the bitmap itself. Width and height are measured in pixels with one pixel equal to one cell on the **bitmap** grid.

Using 'bitmap'

The **bitmap** client displays a variable-size grid, a command panel (on the right), and two "preview" bitmaps. You operate **bitmap** by using mouse buttons to "draw" pixels in the grid, one pixel per cell, and by making selections from the command panel. The preview bitmaps enable you to see how your art work looks in regular and reverse video.

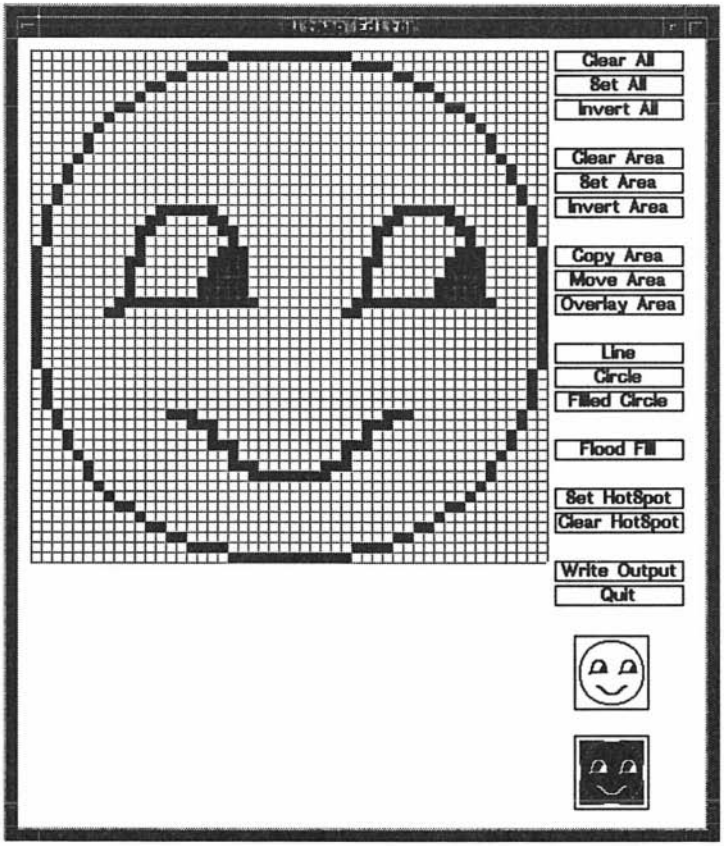


Figure 5-1. The 'bitmap' Client Creates Custom Bitmaps.

Currently, `bitmap` uses the button definitions in the following table:

Table 5-11. Mouse Button Definitions for 'bitmap'.

If you see ...	On a 3-button mouse press ...	On a 2-button mouse press ...
Button 1	The left button.	The left button.
Button 2	The middle button.	Both buttons simultaneously.
Button 3	The right button.	The right button.

The following table shows how to use the grid portion of the `bitmap` window:

Table 5-12. How to Use the 'bitmap' Grid.

If you want to ...	Do this ...
Draw a pixel. Change a cell from background to foreground color.	Click button 1 on that cell.
Invert a pixel color. Change a background colored cell to foreground or a foreground colored cell to background.	Click button 2 on that cell.
Clear a pixel. Change a cell to the background color.	Click button 3 on that cell.

The following table shows how to use the command panel portion of the `bitmap` window:

Table 5-13. How to Use the 'bitmap' Command Panel.

If you want to ...	Click the select button on ...
Set (clear) all cells of the grid to the background color.	Clear All
Set all cells of the grid to the foreground color.	Set All
Set all background colored cells to foreground and all foreground colored cells to background.	Invert All
Set (clear) an area of the grid to the background color.	Clear Area.
Set an area of the grid to the foreground color.	Set Area.
Set any background colored cells in an area of the grid to foreground and any foreground colored cells in that area to background.	Invert Area
Copy one area of the grid to another.	Copy Area
Move an area of the grid to another position.	Move Area
Place one area of the grid over another.	Overlay Area
Draw a line between two points.	Line
Draw a circle with a given center and radius.	Circle
Draw a filled (foreground colored) circle with a given center and radius.	Filled Circle
Fill an enclosed (bounded) area. The area must be completely enclosed.	Flood Fill
Set a "hot spot" to mark the location of the cursor on a cursor bitmap.	Set HotSpot
Erase a "hot spot" from a cursor bitmap.	Clear HotSpot
Save the bitmap to the file specified on the `bitmap` command line.	Write Output
Exit the `bitmap` client.	Quit

Examples

The following examples illustrate some of the possibilities when creating custom bitmaps with `bitmap`.

Creating an Icon Image

You can create a 50 by 50 pixel icon image that you can use for a particular client such as `hpterm` windows. The following bitmap is one example:

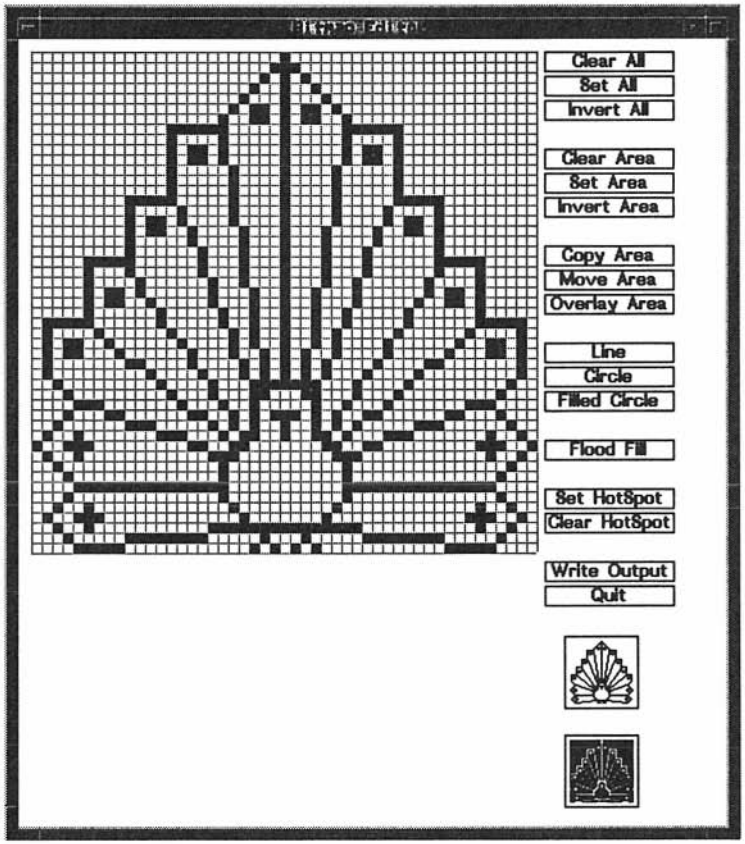


Figure 5-2. A Custom Icon Bitmap.

If you name this bitmap "peacock.bits" and keep it in the `~/bits` directory, where `~` stands for the path to your home directory, you can use the bitmap as

an image for `hpterm` icons by inserting a line similar to the following in your `.Xdefaults` file:

```
Hpwm*HPterm*iconImage:  ~/bits/peacock.bits
```

Whenever you iconify an `hpterm` window, your peacock will display.

Creating Root Window Tiles

You can create tiles of any size with which to pattern your root window. One such pattern is the following:

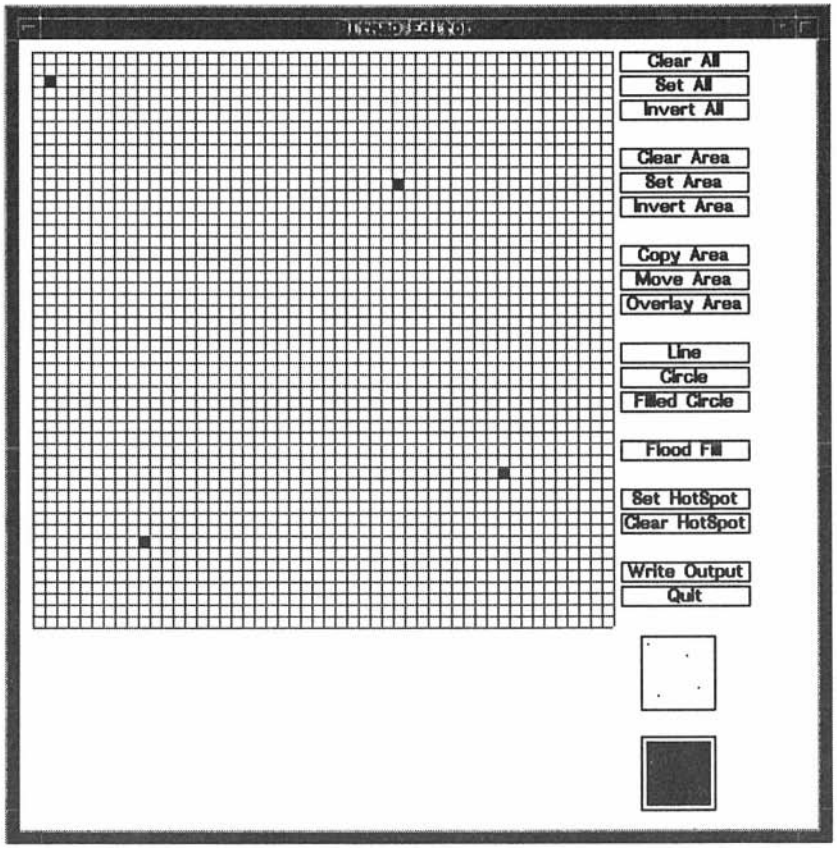


Figure 5-3. A Custom Bitmap for That Spacious Look.

This pattern, called for lack of a better name "space.bits," is a random pattern of foreground-colored pixels. Using the **xsetroot** client described shortly, you can use this bitmap for a truly cosmic effect.

Creating Custom Cursors and Masks

Creating a custom cursor (pointer) requires you to make a cursor bitmap and a cursor **mask** bitmap. The mask provides a background for the cursor and prevents the pixels over which the cursor moves from showing through the cursor bitmap.

For example, because the example above gave you some space to play with, you might want to create the following cursor, named "shuttle.bits" to help you to get from window to window.

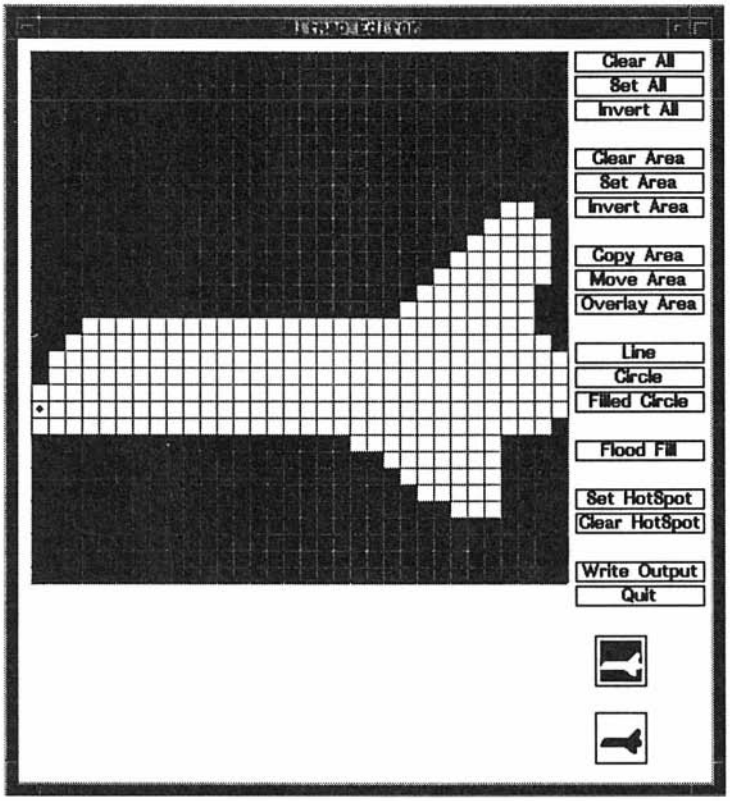


Figure 5-4. A Custom Cursor for Navigating Large Spaces.

Note the **hotspot** at the tip of the shuttle' nose. A hotspot is the single pixel that has been designated as the "point" of the pointer.

The following mask, "mask.bits" is made by inverting the original cursor and adding a few extra lines for shading:

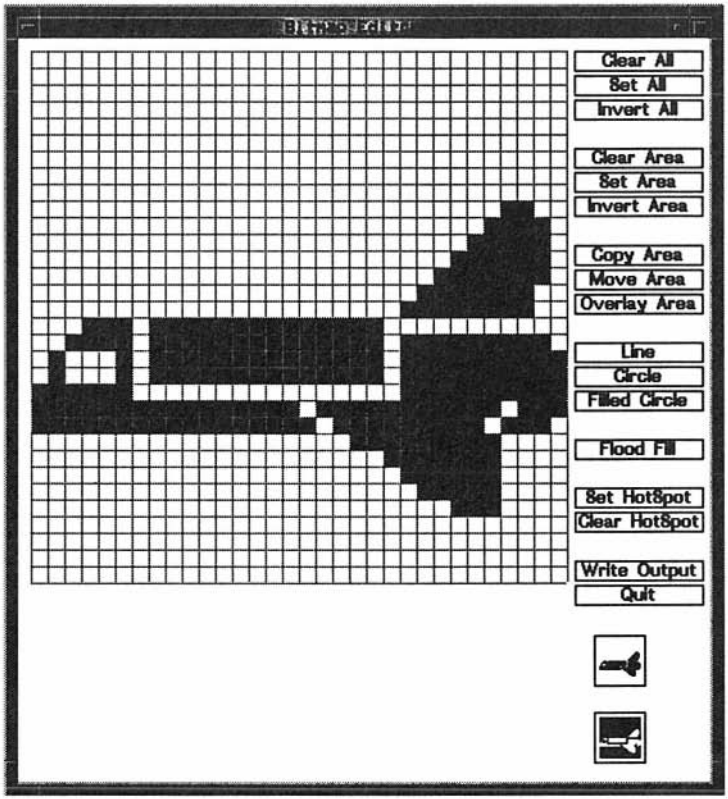


Figure 5-5. A Custom Mask for Navigating Large Spaces.

You employ your custom cursor and mask bitmaps using the **xsetroot** client described next.

Customizing the Root Window with 'xsetroot'

The **xsetroot** client enables you to customize the appearance of the root window. You can add color and pattern to the root window, or modify the shape of the cursor when it's in the root window.

Syntax and Options

The **xsetroot** client has the following syntax:

```
                ┌-help                                  ┐
                |-def                                   |
                |-cursor path/cursor  path/mask         |
                |-bitmap  path/bitmap                   |
                |-mod  x  y                             |
xsetroot        |-gray                                  |
                |-fg  color                             |
                |-bg  color                             |
                |-rv                                    |
                |-solid  color                          |
                └-display  host:display.screen          ┘
```

-help	Prints a summary of the command usage.
-def	Resets unspecified root window attributes to their default values.
-cursor	Specifies the cursor bitmap and mask bitmap to use for the root window cursor.
-bitmap	Specifies a bitmap file with which to tile the root window.
-mod	Specifies a modular grid of dimensions *x* by *y* in the foreground color, making the root window into a plaid pattern.
-gray	Specifies gray (or grey) for the color of the root window.
-fg	Specifies *color* as the foreground color.
-bg	Specifies *color* as the background color.
-rv	Swaps foreground and background colors.

-solid	Specifies the root window should be colored a solid *color.*
-display	Specifies the host, display number and screen number of the root window to change.

Examples

The following examples employ the bitmaps created in the last section.

Changing the Root Window Tile Pattern

To change the tile pattern of the root window to a bitmap such as the "space.bits" bitmap, use the following line:

```
xsetroot -bitmap ~/bits/space.bits
```

This line assumes that you keep your bitmaps in a subdirectory of your home directory called **bitmaps**. The actual **xsetroot** command can be issued either from the command line once you've started X or from a line in your **.x11start** file (in which case the changes are made as X11 starts).

Changing the Root Window Cursor

To change the shape of the root window cursor to a bitmap such as the "shuttle.bits" bitmap created above, use the following line:

```
xsetroot -cursor ~/bits/shuttle.bits ~/bits/mask.bits
```

Again, you can issue this line either at the command-line prompt once you've started X or include it as part of your **.x11start** file. Remember, the ~ signifies the path to your home directory.

Working with Fonts

The X Window System includes a variety of **fonts.** A font is a type style, that is, a style in which text characters are printed. For example, the text of most newspapers is printed in the Times Roman font while the headlines are usually printed in Helvetica.

You will find a complete list of valid font names in the `/usr/lib/X11/fonts` directory. Use the following HP-UX command to list the names:

```
ls -x /usr/lib/X11/fonts Return
```

Although font names have extensions, usually a `.snf` (server natural format) or `.scf` (server compressed format), you don't have to type the extension when you specify a font.

Choosing Where to Specify a Font

Usually, you specify fonts in the `.Xdefaults` file in your home directory. However, you can specify the font of an individual client (such as the text of a single window) in the command line that starts the client. If you start the client when you start X11, the command line will be in the `.x11start` file. If you start the client from a menu, the command line will be in the `.hpwmrc` file.

Making All Instances of a Client Have the Same Font

By inserting a command line in the `.Xdefaults` file in your home directory, you can make every instance of a particular client have the font that you specify.

The syntax for the line is as follows:

client`*font:` *fontname*

The following line in your `.Xdefaults` file changes the font of *every* `hpterm` window to hp8.8×16.

```
HPterm*font: hp8.8×16
```

Note that your `.Xdefaults` file may already contain a line specifying the font for a client, so you only need to change the name of the font.

Specifying the Font of a Window that Starts Automatically

The following line, which uses the standard command-line syntax, in your `.x11start` file overrides any font specification in the `.Xdefaults` file and creates *this particular* `hpterm` window with an hp8.8×16 font.

```
hpterm -fn hp8.8×16 &
```

Specifying the Font of a Window that Starts from a Menu

The following line, which uses the standard menu selection syntax, in your `.hpwmrc` file overrides any font specified in the `.Xdefaults` file and, when you choose the **New Window** selection from the menu, creates an **hpterm** window with a font of hp8.8×16.

```
"New Window"  f.exec "hpterm -fn hp8.8×16 &"
```

Choosing a Font to Specify

The following table lists X Window System fonts. For best results, type the font name exactly as it appears below.

Table 5-14. X11 Fonts*.

Fonts						
6x10	6x12	6x13	8x13	8x13bold	9x15	9x16apl
9x16bas	9x21apl	9x21bas	9x21ibm	12x21apl	12x21bas	12x28apl
12x28bas	12x28ibm	a14	apl-s25	calc.12x16	calc.6x8	chp-s25
chs-s50	cr.12x20	cr.12x20b	cursor	cyr-s25	cyr-s30	cyr-s38
fcor-20	fg-13	fg-16	fg-18	fg-20	fg-22	fg-25
fg-30	fg-40	fg1-25	fgb-13	fgb-25	fgb1-25	fgb1-30
fgi-20	fgi1-25	fgs-22	fixed	fqxb-25	fr-25	fr-33
fr1-25	fr2-25	fr3-25	frb-32	fri-33	fri1-25	ger-s35
grk-s25	grk-s30	hbr-s25	hbr-s40	hp8.10x20	hp8.10x20b	hp8.12x15
hp8.6x13	hp8.6x13b	p8.6x8	hp8.6x8b	hp8.7x10	hp8.8x16	hp8.8x16b
hp8.8x16i	ipa-s25	iso1.13	iso1.13b	iso1.15	iso1.16	iso1.16b
iso1.20	iso1.20b	iso1.8	k14	kana.10x18	kana.10x20	kana.12x24
kana.8x16	kana.8x18	kana14	krivo	lat-s30	line.8x16	math.18x30
math.6x8	math.8x16	met25	micro	oldera	pica.18x30	plunk
r14	rot-s16	sans12	sansb12	sansi12	serif10	serif12
serifb10	serifb12	serifi10	serifi12	sub	subsub	sup
supsup	swd-s30	sym-s25	sym-s53	variable	vbee-36	vctl-25
vg-13	vg-20	vg-25	vg-31	vg-40	vgb-25	vgb-31
vgbc-25	vgh-25	vgi-20	vgi-25	vgi-31	vgl-40	vgvb-31
vmic-25	vr-20	vr-25	vr-27	vr-30	vr-31	vr-40
vrb-25	vrb-30	vrb-31	vrb-35	vrb-37	vri-25	vri-30
vri-31	vri-40	vsg-114	vsgn-57	vshd-40	vtbold	vtsingle
vxms-37	vxms-43	xif-s25				
*Font names are shown without extensions.						

Displaying a Font with 'xfd'

You can display the complete character set of any valid X Window System font using the `xfd` client.

Syntax and Options

The syntax for **xfd** is as follows:

xfd [**-rv** | **-fg** *color* | **-bg** *color* | **-bf** *font* | **-tl** *title* | **-in** *icon* | **-icon** *path/bitmap* | **-verbose** | **-gray** | **-start** *charnumber* | **-geometry** *parameters* | **-display** *host:display.screen*] [*fontname*]

-rv	Switches the foreground and background colors (reverse video).
-fg	Specifies the foreground color for **xfd**.
-bg	Specifies the background color for **xfd**.
-bf	Specifies *font* as the font to use for displaying messages at the bottom of the **xfd** window.
-tl	Specifies *title* as the title that should appear in the title bar of the **xfd** window frame.
-in	Specifies *icon* as the name to use for the icon label when an **xfd** client is iconified.
-icon	Specifies the path and filename of the bitmap to use as the icon for the **xfd** client.
-verbose	Displays additional information about a character including: left bearing, right bearing, ascent, descent, and width.
-gray	Specifies a gray background.
-start	Specifies that character number *charnumber* should be the first character displayed (the character in the upper left corner).

-geometry	Specifies the size (width×height) and location (±column±row) of an **xfd** window.
-display	Specifies the host, display number, and screen number on which to display **xfd**.
font	Specifies the font to display. If an invalid font, or no font is used, the **fixed** font is displayed by default.

Using 'xfd'

The **xfd** client creates a 16 by 16 grid by default, but you can change the size using the **-geometry** option. Each cell of the grid, starting at the upper left corner, contains a character of the font named on the command line.

Currently, **xfd** uses the button definitions in the following table:

Table 5-15. Mouse Button Definitions for 'xfd'.

If you see ...	On a 3-button mouse press ...	On a 2-button mouse press ...
Button 1	The left button.	The left button.
Button 2	The middle button.	Both buttons simultaneously.
Button 3	The right button.	The right button.

Use the following actions to operate the **xfd** client:

Table 5-16. Using the 'xfd' Client.

If you want to ...	Do this ...
Page forward to see characters from the specified font that are not currently displayed.	Position the pointer on the **xfd** window and click button 3.
Page backward to see the previously displayed characters.	Position the pointer in the **xfd** window and click button 1.
Display the character set starting with a particular character.	Use the **-start** *charnumber* option on the command line when you start **xfd**.
Show the decimal and hexadecimal value of a character.	Position the pointer in the grid for that character and click button 2.
Show additional information about a character set including left bearing, right bearing, ascent, descent, and width.	Use the **-verbose** option on the command line when you start **xfd**.

Example

The following command line starts an **xfd** window displaying the met25 (met25.scf) font in verbose mode. The name of the font appears as a reminder in the title bar.

```
xfd -verbose -tl met25 -geometry 300x300-1-1 met25 & Return
```

The window has a 300 by 300 pixel size and appears in the lower right corner of the screen.

The result of issuing this command line is as follows:

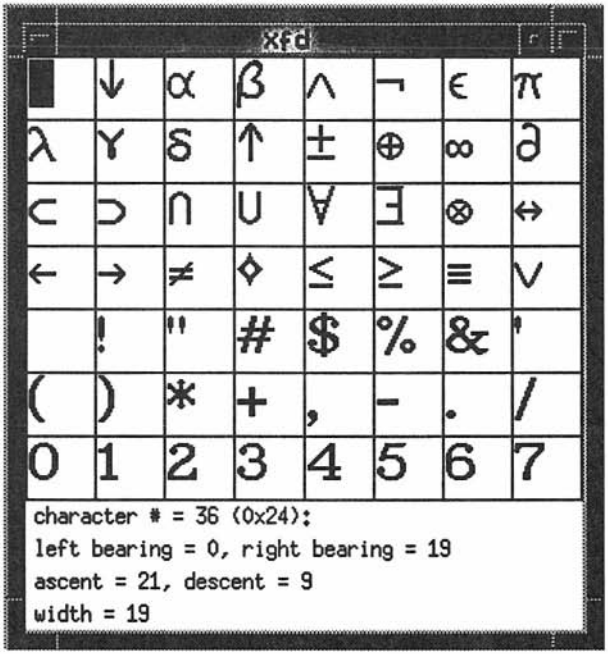


Figure 5-6. The First Page of 'met25.scf'.

The small size of the geometry, combined with the large size of the font, prohibits the entire character set from displaying on the first "page" of the grid. You can view the remaining characters of the font by positioning the pointer anywhere in the window and clicking button 3.

You can display information about a character by positioning the pointer on that character and clicking button 2. The figure above shows information for the $ character (charnumber = 36) at the bottom of the window.

Using Remote Hosts

Part of the potential of the X Window System is that it enables you to be in two places at once—sort of. You can be logged into your local system working locally and, at the same time be logged into one or more remote hosts.

Gaining Access to Remote Hosts

To gain access to a remote host, you must have the following:

- The address and hostname of the remote host listed in your system's `/etc/hosts` file.
- A valid login (username and password) and home directory on the remote host.
- The hostname of the remote host listed in a `/etc/X0.hosts` file on your system.
- Your system listed in a `.rhosts` file in your home directory *on the remote host.*
- The hostname of the remote host listed in a `.rhosts` file in your home directory *on your local system.*

Setting Up a Login on a Remote Host

To set up a login on a remote host, you need to check that the remote host has a valid internet address and hostname in your system's `/etc/hosts` file, the file that tells the system the address of the other systems on the network.

Also, you need to talk to the system administrator for the remote system. You will need a username, password (if necessary), and a home directory on that system. That way, when you log into the remote host, the remote host will know who you are and where you belong in the directory structure.

Setting Up an 'X0.hosts' File

The remote host must have permission to connect to your display server and display a client program. It gets this permission by being listed in the `/etc/X0.hosts` file on your system. The `X0.hosts` file is an ASCII file that contains the hostnames of all remote hosts that have permission to use your

server to display clients on your display screen. Each hostname occupies a separate line as follows:

host1
host2
host3

You can create the file for yourself using any ASCII text editor, or you can use the **xhost** client described below to dynamically add or delete hosts. Changes made with **xhosts** are in effect only for the length of your X session.

Note that the "0" of **X0.hosts** signifies a particular display (combination of screen, keyboard, and mouse) on your system. This is typically the console. If you have another display configuration, you may need another host file. For example, if you are the second display on a system, your host file would probably be **X1.hosts**. A "display" can be either physical (for example, display 0 could correspond to seat 0) or "logical" (for example, if you switch between several configurations, your display could have several logical display numbers, one for each different configuration). For more information, see chapter 7.

Preparing a '.rhosts' File

A **.rhosts** file, placed in your home directory, enables any remote host listed in the file to connect to your system using your login account without having to go through the drudgery of formally logging in and giving a password.

Although this may be a convenience to you, it may be an undue opportunity to someone else.

Note

Depending on your situation, a **.rhosts** file could undermine the security of your system and other systems on the network. Check with your system administrator and analyze the security needs of your situation to develop an appropriate plan.

The **.rhosts** file is an ASCII file containing one remote host per line as in the following syntax:

host1
host2
*host**

To create a **.rhosts** file, you should be in your home directory. Use the following steps:

1. Start your editor and open a file called **.rhosts**.
2. Type the name of the remote host that you want to add.
3. Press **Return** to move to the next line.
4. Repeat steps 2 and 3 for each remote host you want to add.
5. Check your spelling, save the file, and exit your editor.

The **.rhosts** file assumes that your remote username is the same as your local user name.

Adding and Deleting Hosts with 'xhost'

The **xhost** client provides you with a convenient way to dynamically control access to your local system. Using **xhost**, you can add or delete a remote host's permission to access your local X11 display server.

Note that **xhost** only adds or deletes a remote host to or from an internal list created at the start of an X session. It does *not* change the **/etc/X0.hosts** file. To permanently add or subtract access permission you must edit the **/etc/X0.hosts** file using an ASCII text editor such as **vi**.

Syntax and Options

The syntax for **xhost** is as follows:

```
        [+]host
        -host
xhost   +
        -
        (no option)
```

host — Adds *host* to the list of remote hosts with permission to access your local X server.

+host — Adds *host* to the list of remote hosts with permission to access your local X server.

-host	Deletes *host* from the list of remote hosts with permission to access your local X server.
+	Turns off access control, allowing any remote host to access your local X server.
–	Restricts access to your local X server to remote hosts currently listed in your local **/etc/X0.hosts** file.
no option	Prints the list of remote hosts that currently have access to your X server.

You can run **xhost** from the command line at any time you need to change access to your server or to see the current list of remote hosts with access to the server. Remember, changes you make using **xhost** are temporary. They last only as long as your current X session.

Example

The following example allows the remote host **hpcvfgg** to access your local display. As soon as you quit the window system, the access permission is revoked.

```
xhost +hpcvfgg Return
```

Starting Programs on a Remote Host

The "Starting Programs" section of chapter 4 covered starting remote clients and non-clients from the command line. You can, however, start remote programs without typing a lengthy command after the command-line prompt.

Starting a Remote Program when you start X11

One way to start a remote program, either a client or non-client, is to start the program when you first start X11. This enables you to have the remote program as a part of your initial environment.

To start a remote client when you start X11, you need to edit the **.x11start** file in your home directory to include one line for each remote client you want to start. The lines are similar to the following:

remsh *host* **-n** *client* **-display** *host:display.screen* [&]

Here *host* is the name of the remote host. The *client* can be any X client. And the **-display** option specifies the system, display number, and screen number where the client is to display, typically your local system.

To start a remote non-client when you start X11, edit your **.x11start** file to include one line for each remote non-client. The line begins by starting a remote shell (**remsh**), then a terminal emulation window in whiich to run the non-client, and finally the non-client:

remsh *host* **-n hpterm -display** *host:display.screen* **-e** *non-client* [&]

The **-e** option ("e" for execute), when used with an **hpterm** or **xterm** client, executes a command, in this case the non-client.

Note that an alternate syntax is to start an **hpterm** window and use the **-e** option to execute a remote login (**rlogin**) that makes the window a terminal of the remote host.

For example, the following lines start a remote login (non-client) and a remote load histogram (client) on the host **hpcvfaa** and display the results on the console of the local system, **hpcvfbb**:

```
remsh hpcvfaa -n /usr/bin/X11/xload -display hpcvfbb:0.0 &
hpterm -title "hpcvfaa login" -e rlogin hpcvfaa &
```

Starting a Remote Program from a Menu

Starting a remote program from a menu requires editing the **.hpwmrc** to include the proper line to start the program. The process is similar to starting the program from **.x11start**.

Use a line similar to the following to start a remote client:

selection **f.exec "remsh** *host* **-n** *client* **-display** *h:d.s* **&"**

To start a remote non-client, use the above syntax, adding a **-e** option as the last option before the **&**. Alternately, create an **hpterm** window and use **-e rlogin** *host* to start a remote login.

The explanation of this syntax is the same as the syntax used in **.x11start** with the exception of *selection,* the selection that appears on the menu, and **f.exec**, the HP Window Manager function that starts a process, in this case an **hpterm** window.

Example

The following example starts a login on remote host `hpcvfaa`. The login process is initiated by choosing the "hpcvfaa Login" selection from the root menu.

```
# Root Menu Description
Menu  DefaultRootMenu
{
"Root Menu"     f.title
"New Window"    f.exec "hpterm &"
"hpcvfaa Login" f.exec "hpterm -e rlogin hpcvfaa &"
"Shuffle Up"    f.circle_up
"Shuffle Down"  f.circle_down
"Refresh"       f.refresh
no-label        f.separator
"Restart"       f.restart
}
```

Where To Go Next

This chapter has discussed customizing the operation of your window system environment to suit your personal needs. There is additional customization that you can perform beyond what was presented here. Some of it is a little more difficult to comprehend and it would be a good idea to consult with your system administrator before attempting to implement some of the changes.

If you are satisfied with the current look and performance of your window system environment, you may want to stop here, use the system for a few days or weeks, and then perhaps "fine tune" it based on your experience.

On the other hand, if you are interested in more extensive customizations to the HP Window Manager, in special environment configurations, in printing, or in graphics, you should read chapters 6, 7, 8, and 9 respectively.

If your interest is in programming, turn to one of the programming manuals.

6

Managing Windows

Managing windows is the job of the window manager. This chapter begins by briefly mentioning the clients related to window management. But most of the chapter discusses the nitty-gritty details of how to use the HP Window Manager, its resources, and functions to manage your window environment.

It is *not* necessary to read this chapter to use the window manager or X, but if your management needs go beyond adding and deleting menu selections, browsing this chapter should prove interesting. After discussing the clients, the chapter reviews some familiar aspects of window control, but becomes more technical once these basics have been covered.

The chapter organizes window manager resources and functions into the following task-oriented topics:

- Managing the general appearance of window frames.
- Working with icons.
- Managing window manager menus.
- Using the mouse.
- Using the keyboard.
- Controlling window size and placement.
- Controlling resources with focus policies.
- Matting clients.

Clients That Help You Manage Windows

Of the clients listed in the reference section of this manual, five are directly related to window management:

- `resize`
- `xrefresh`
- `xwininfo`
- `uwm`
- `hpwm`

Resetting Environment Variables with 'resize'

The `resize` client resets three environment variables: TERM, LINES, and COLUMNS. This enables a shell to reflect the current size of its window.

Don't confuse `resize`, the client, with `f.resize` the window manager function. The `f.resize` function changes the size of a window, but does not reset any environment variables. The `resize` client, on the other hand, does not change the size of a window, but it does reset the environment variables. Resetting the environment variables enables non-client programs to adjust their output to the window's new size.

When to Use 'resize'

Use `resize` whenever you resize a window and want a non-client program running in that window to reflect the window's new size. The `resize` client is typically used as an argument to the HP-UX `eval` command.

Syntax and Options

The syntax for `resize` is as follows:

`resize` { `-u` | `-c` }

-u Resets the environment variables for `sh` and `ksh` shells.

-c Resets the environment variables for `csh` shells.

Example

To see what the current COLUMN and LINES settings are, type the following command:

```
resize Return
```

After you have resized a window either by dragging the window frame or by choosing the "Size" selection from the system menu, you can reset the LINES, and COLUMN environment variables to reflect the new window size by issuing the following command:

```
eval `resize` Return
```

If you find yourself typing the above command too often, you can make things a little easier on yourself. If you use **csh**, try using an alias. The following line in your **.aliasbin** file enables you to run **resize** by typing **xr**.

```
alias xr 'set noglob; eval `resize`'
```

If you use **sh** or **ksh** create an **xr** function like the following:

```
xr()  {eval `resize`;}
```

Repainting the Screen with 'refresh'

The **xrefresh** client "repaints" the screen or a specified portion of the screen. It does this by mapping, then immediately unmapping, a window over the area to be repainted. This obscuring-unobscuring causes the area to be redrawn. Repainting a screen removes the "graphics litter" that occasionally disfigures a screen.

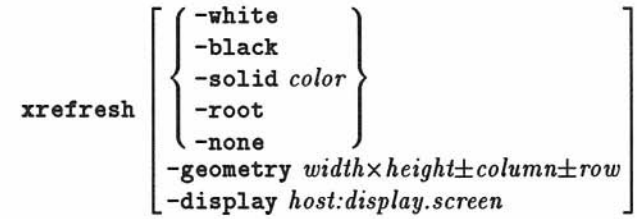
The **xrefresh** client performs a similar task to the **f.refresh** window manager function. However, the **xrefresh** client, because of its options, is more versatile.

When to Use 'xrefresh'

You can use **xrefresh** from the command line of any terminal window and, using the **-display** option, you can repaint any display.

Syntax and Options

The syntax for **xrefresh** is as follows:



-white	Uses a white window to map the screen.
-black	Uses a black window to map the screen.
-solid	Uses a *color* colored window to map the screen.
-root	Uses the root window to map the screen.
-none	Uses a transparent window to map the screen (default).
-geometry	Repaints a *width*×*height* region located at ±*column*±*row* on the screen (dimensions are in pixels).
-display	Specifies the screen to refresh.

Example

The following example illustrates using **xrefresh** from the command line to repaint the upper left quarter of the screen.

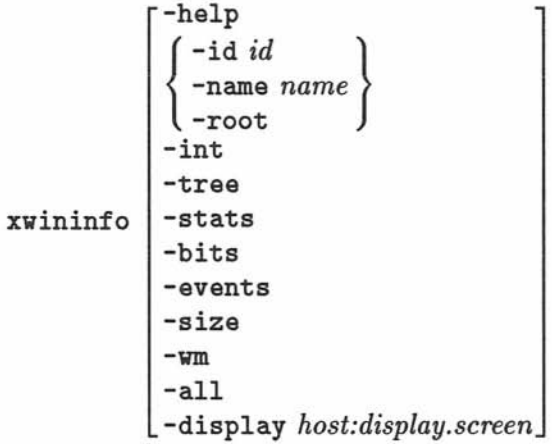
```
xrefresh -white -geometry 800x400+1+1
```

Getting Window Information with 'xwininfo'

The **xwininfo** client is a utility program that displays useful information about windows.

Syntax and Options

The syntax for **xwininfo** is as follows:



-help	Prints a summary of the command usage.
-id	Specifies the target window by window id.
-name	Specifies the target window by name.
-root	Specifies the root window as the target.
-int	Displays window id (as decimal), location, size, depth, and other information.
-tree	Displays ids and names of the root, parent, and child windows.
-stats	Displays window id (as hexadecimal), location, size, depth, and other information.
-bits	Displays information about bit and storage attributes.
-events	Displays event masks of the target window.
-size	Displays sizing information about the target window.
-wm	Displays the window manager hints for the target window.
-all	Displays all available information about a window.
-display	Specifies the host, display, and screen to target.

Example

This example illustrates the result of issuing the following command:

```
xwininfo -stats Return
```

Once you issue the command, select a window as the target of your inquiry by moving the pointer into that window and clicking the select button.

```
xwininfo ==> Window id: 0x10007f (has no name)
  ==> Upper left X: 651
  ==> Upper left Y: 350
  ==> Width: 626
  ==> Height: 653
  ==> Depth: 8
  ==> Border width: 0
  ==> Window class: InputOutput
  ==> Window Map State: IsViewable
```

Managing Windows with 'uwm'

The **uwm** client provides an alternative to managing windows with the HP Window Manager. Windows managed with **uwm** have variable width borders but do not have the functional window frame or 3-D appearance of **hpwm**-managed windows.

The **uwm** window manager uses a single generic menu displayed on the root window to control the size, location, iconification, and other basic operations of the objects on the root window. Like **hpwm**, **uwm** allows the moving and resizing of windows without recourse to a menu.

The **uwm** client starts from **/usr/lib/X11/sys.x11start** or **$HOME/.x11start** as part of the start procedure after you issue the **x11start** command. It receives its initial configuration information from the file **$HOME/.uwmrc**, if that file is available, or from **/usr/lib/X11/uwm/system.uwmrc** if it isn't. If no configuration file can be found, built-in default values are used.

When to Use 'uwm'

You can use **uwm** as an alternative to **hpwm**. Typically, you edit the **.x11start** file in your home directory so that **uwm** is the first client started when you start X11.

Syntax and Options

The syntax for **uwm** is as follows:

```
uwm [-f filename
     -display host:display.screen]
```

-f — Names a file other than **.uwmrc** from which to read initial configuration information.

-display — Specifies the screen to use.

Example

You can call **uwm** from **.x11start** in your home directory by making the following two modifications:

```
hpwm $@ &     Delete this line, or comment it out
uwm $@ &      Insert this line
```

Managing Windows with the HP Window Manager

The Hewlett-Packard Window Manager (**hpwm**) is an X11 client that manages the appearance and behavior of objects on the root window. You control **hpwm** and its management operations using a mouse, a keyboard, and a functional window frame similar to Microsoft's Presentation Manager. Additionally, **hpwm** has a root menu to assist you in the general control of the root window.

The **hpwm** client receives configuration information from three files: **/usr/lib/X11/sys.Xdefaults**, **/usr/lib/X11/system.hpwmrc**, and **/usr/lib/X11/app-defaults/Hpwm**. You can copy the first two of these files to your home directory, as **.Xdefaults** and **.hpwmrc** respectively, and edit them to create a window manager that exactly fits your needs.

How to create your own personal window manager is the subject of the rest of this chapter.

When to Use 'hpwm'

The HP Window Manager is the default window manager for your X Window System. It is started from **$HOME/.x11start** when you start X11. If that file doesn't exist, **hpwm** is started from **/usr/lib/X11/sys.x11start**.

Syntax and Options

The syntax for **hpwm** is as follows:

hpwm [**-display** *host:display.screen*
-xrm *resourcestring*]

-display Specifies the screen to use.

-xrm Specifies using the named resource on starting.

Example

The following line in **.x11start** in your home directory starts **hpwm**.

```
hpwm $@ &
```

The **$@** passes the window manager options specified on the **x11start** command line.

Managing the General Appearance of Window Frames

In chapter 5, you read about **/usr/lib/X11/sys.Xdefaults** and **.Xdefaults**.

The **sys.Xdefaults** file is the system file that controls the X environment of users who don't have a **.Xdefaults** file in their home directory. **.Xdefaults** overrides the system-wide effects of **sys.Xdefaults**, enabling you to customize your own environment while not interfering with the environments of others.

By editing **.Xdefaults**, you can control the general appearance of the window frames in your environment. If you are a system administrator, you can control the system-wide general appearance of window frames by editing **/usr/lib/X11/sys.Xdefaults**.

Three aspects of the general appearance of window frames are under your control.

Color The color of foreground, background; and top, bottom, and side shadows.

Tile	The mixture of foreground and background color that composes the pattern of the frame surface.
Font	The style (including size) of the text characters in the title bar, menus, and icon labels.

To control color, tile pattern, or fonts, you specify a value for the appropriate window manager **resource.** A resource controls an element of appearance or behavior. Resources are always named for the elements they affect.

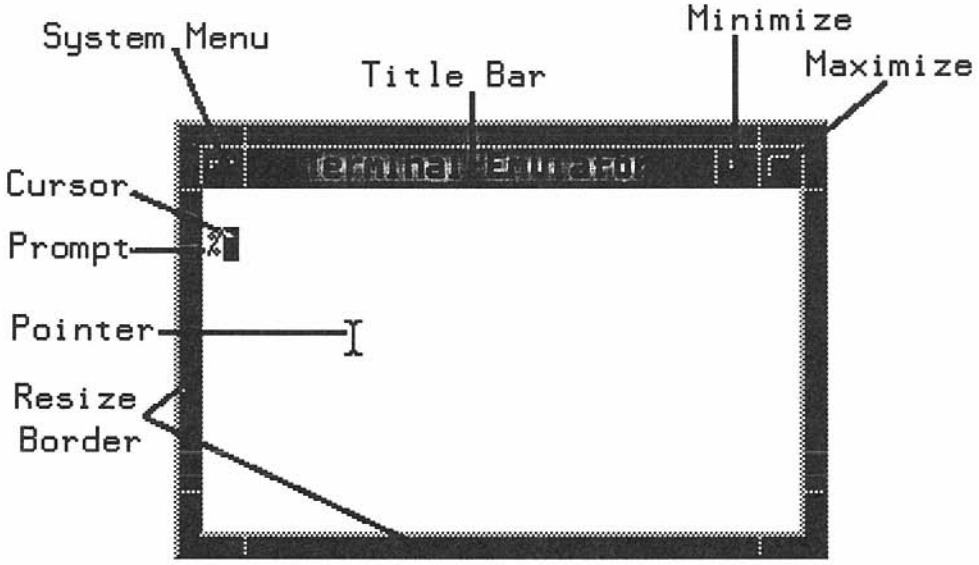


Figure 6-1. An HP Window Manager Frame Showing Frame Elements.

For example, suppose you want to color the background of your window frame (an element of appearance) Firebrick red. You would edit `.Xdefaults` making `Hpwm*background:` (the resource controlling the background color of the frame) the color Firebrick (a color value). The line in `.Xdefaults` would read as follows:

```
Hpwm*background: Firebrick
```

Coloring Window Frames

You can use any of the standard X11 colors listed in `/usr/lib/X11/rgb.txt` to color frame elements. Additionally, you can create your own colors using hexadecimal values. Frame elements and resources exist for inactive windows (any window not having the current keyboard focus) and for the active window (the window having the current keyboard focus). This enables you to distinguish the active window by giving it special "active window" colors.

Coloring Individual Frame Elements

The following table lists the individual elements of inactive and active window frames, and the resources that control their color.

Table 6-1. Coloring Window Frames with HP Window Manager Resources.

To color this ...	Use this resource ...	The default value is ...
Background of inactive frames.	`background`	white
Left and upper bevel of inactive frames.	`topShadowColor`	white
Right and lower bevel of inactive frames.	`bottomShadowColor`	black
Foreground (title bar text) of inactive frames.	`foreground`	black
Background of the active frame.	`activeBackground`	background
Left and upper bevel of the active frame.	`activeTopShadowColor`	topShadowColor
Right and lower bevel of the active frame.	`activeBottomShadowColor`	bottomShadowColor
Foreground (title bar text) of the active frame.	`activeForeground`	foreground

Coloring Frame Elements Automatically

Additionally, two resources exist that enable you to circumvent specifying the individual colors of each and every frame element.

`makeColors`	Uses the background color of the inactive window to generate colors for the other inactive window frame elements, giving the frame a 3-D look.
`makeActiveColors`	Uses the background color of the active window to generate colors for the other active window frame elements, giving the frame a 3-D look.

Exactly which elements you color with `makeColors` and `makeActiveColors` depends on which of three values you give the resources. The resource values and their effect on frame elements is as follows:

Table 6-2. The Values to Use for Automatically Coloring Frame Elements.

To color these elements ...	Use this value ...
All bevel elements and the foreground (tile elements for top and bottom shadows are set to foreground).	`all`
All bevel elements.	`shadow`
No elements automatically colored.	`none`

Example

For example, the following lines in the `.Xdefaults` file in your home directory give the window manager frame a maroon foreground and a gray background. The `makeColors` line uses the background color to generate colors for the top and bottom shadow elements so that a pleasing 3-D effect is achieved.

```
Hpwm*foreground: Maroon
Hpwm*background: Gray
Hpwm*makeColors: shadow
```

Changing the Tile of Window Frames

A **tile** is a rectangle that provides a surface pattern or a visual texture. The concept is analogous to the use of ceramic tiles to provide a floor or countertop with a pattern or texture.

Generally, the fewer the number of colors your display can produce, the more important tiling will be to you. This is because tiling provides you with a way to "mix" foreground and background colors into a third color "pattern."

For example, if you had a monochrome display (two colors—black and white), you could tile the window frames of your X environment in shades of gray to achieve a pleasing 3-D look.

The HP Window Manager has resources that enable you to tile the frame background and bevels for both inactive and active windows.

Table 6-3. Tiling Window Frames with Window Manager Resources.

To tile this ...	Use this resource ...	The default value is ...
Background of inactive frames.	`backgroundTile`	25_foreground
Right and lower bevels of inactive frames.	`bottomShadowTile`	foreground
Left and upper bevels of inactive frames.	`topShadowTile`	50_foreground
Background of the active frame.	`activeBackgroundTile`	foreground
Right and lower bevels of the active frame.	`activeBottomShadowTile`	bottomShadowTile
Left and upper bevels of the active frame.	`activeTopShadowTile`	topShadowTile

The following table lists the acceptable values for tile resources:

Table 6-4. The Values to Use for Tiling Window Frames.

To tile an element this color ...	Use this value ...
The foreground color.	foreground
The background color.	background
A mix of 25% foreground to 75% background.	25_ foreground
A mix of 50% foreground to 50% background.	50_ foreground
A mix of 75% foreground to 25% background.	75_ foreground
In horizontal lines alternating between the foreground and background color.	horizontal_ tile
In vertical lines alternating between the foreground and background color.	vertical_ tile
In diagonal lines slanting to the right alternating between the foreground and background color.	slant_ right
In diagonal lines slanting to the left alternating between the foreground and background color.	slant_ left

The following figure illustrates the valid tile values:

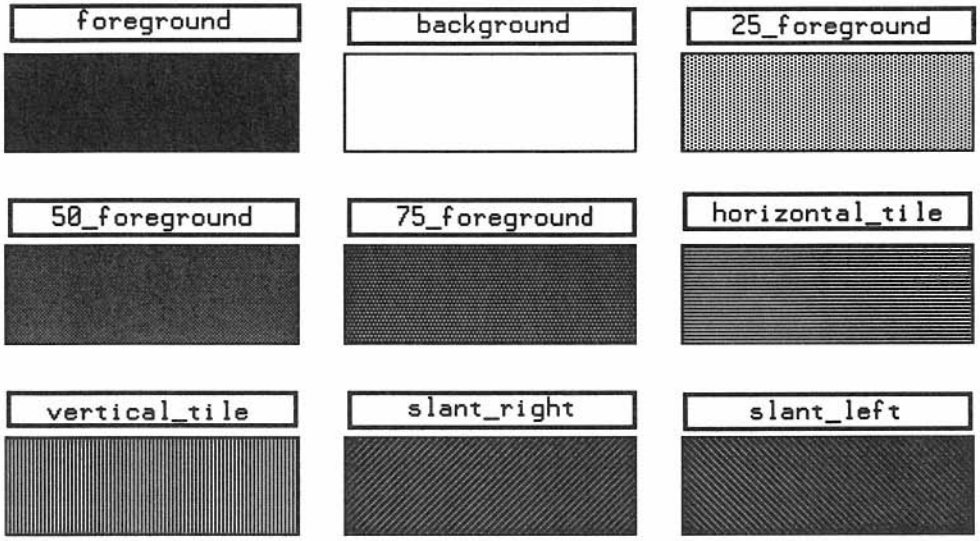


Figure 6-2. Valid Tile Values.

Specifying a Different Font for the Window Manager

The default font for the text of the HP Window Manager is the **fixed** font. However, you can use the **font** resource to specify a different font if you desire. The **font** resource can use any valid X11 font as its value. Valid fonts are contained in the **/usr/lib/X11/fonts** directory.

The following table lists the valid X11 fonts.

Table 6-5. Valid X11 Fonts*.

Fonts						
6x10	6x12	6x13	8x13	8x13bold	9x15	9x16apl
9x16bas	9x21apl	9x21bas	9x21ibm	12x21apl	12x21bas	12x28apl
12x28bas	12x28ibm	a14	apl-s25	calc.12x16	calc.6x8	chp-s25
chs-s50	cr.12x20	cr.12x20b	cursor	cyr-s25	cyr-s30	cyr-s38
fcor-20	fg-13	fg-16	fg-18	fg-20	fg-22	fg-25
fg-30	fg-40	fg1-25	fgb-13	fgb-25	fgb1-25	fgb1-30
fgi-20	fgi1-25	fgs-22	fixed	fqxb-25	fr-25	fr-33
fr1-25	fr2-25	fr3-25	frb-32	fri-33	fri1-25	ger-s35
grk-s25	grk-s30	hbr-s25	hbr-s40	hp8.10x20	hp8.10x20b	hp8.12x15
hp8.6x13	hp8.6x13b	p8.6x8	hp8.6x8b	hp8.7x10	hp8.8x16	hp8.8x16b
hp8.8x16i	ipa-s25	iso1.13	iso1.13b	iso1.15	iso1.16	iso1.16b
iso1.20	iso1.20b	iso1.8	k14	kana.10x18	kana.10x20	kana.12x24
kana.8x16	kana.8x18	kana14	krivo	lat-s30	line.8x16	math.18x30
math.6x8	math.8x16	met25	micro	oldera	pica.18x30	plunk
r14	rot-s16	sans12	sansb12	sansi12	serif10	serif12
serifb10	serifb12	serifi10	serifi12	sub	subsub	sup
supsup	swd-s30	sym-s25	sym-s53	variable	vbee-36	vctl-25
vg-13	vg-20	vg-25	vg-31	vg-40	vgb-25	vgb-31
vgbc-25	vgh-25	vgi-20	vgi-25	vgi-31	vgl-40	vgvb-31
vmic-25	vr-20	vr-25	vr-27	vr-30	vr-31	vr-40
vrb-25	vrb-30	vrb-31	vrb-35	vrb-37	vri-25	vri-30
vri-31	vri-40	vsg-114	vsgn-57	vshd-40	vtbold	vtsingle
vxms-37	vxms-43	xif-s25				
*Font names are shown without extensions.						

The Syntax for Declaring Resources

The above general appearance resources for the HP Window Manager and their values are specified in `sys.Xdefaults` (system-wide) or `.Xdefaults` (your personal environment). The syntax you use differs depending on whether you want the resource to control the general appearance of an element or the general appearance of that element *for a particular object.*

For example, the syntax you use to specify a frame background of Wheat is different from the syntax you use to specify that only menus have a background of Wheat.

The Syntax for the General Appearance of Elements

Use the following syntax in `sys.Xdefaults` or `.Xdefaults` to specify the general appearance of frame elements:

`Hpwm*`*resource*`:` *value*

For example, if you want the foreground and background of inactive window frames to be the opposite of the foreground and background of the active window frame, and you choose the colors SteelBlue for background and VioletRed for foreground, you would have the following lines in your `.Xdefaults` file.

```
Hpwm*makeColors:          none
Hpwm*background:          SteelBlue
Hpwm*foreground:          VioletRed
Hpwm*activeBackground:    VioletRed
Hpwm*activeForeground:    SteelBlue
```

The Syntax for Window Frame Elements of Particular Objects

You can specify the general appearance of window frame elements for three particular objects.

- Menus (includes *both* system and root menus).
- Icons (includes the frame elements of *all* icons).
- Clients (includes the frame elements of *all* clients).

This gives you the ability to select a different color or font for a particular object, perhaps menus, while the other objects (icons and fonts) remain the same. To do this, use the following syntax:

`Hpwm*{menu | icon | client}*`*resource*`:` *value*

For example, if you want the general appearance of the clients in your environment to be as above, SteelBlue and VioletRed, but want your menus to be different, you could add the following lines to `.Xdefaults`.

```
Hpwm*makeColors:          none
Hpwm*background:          SteelBlue
Hpwm*foreground:          VioletRed
Hpwm*activeBackground:    VioletRed
Hpwm*activeForeground:    SteelBlue

Hpwm*menu*background:     SkyBlue
Hpwm*menu*foreground:     White
```

Working with Icons

Icons provide a handy way to straighten up a cluttered workspace. They also provide you with a great tool for efficient multi-processing. For example, you could open several windows, start processes in each, and then iconify them all letting the processes run their individual courses while you sit back and read your electronic mail and work in an editing window.

Studying Icon Anatomy

Icons consist of two parts. Like the other objects that appear on the root window, you can configure the appearance of all icons in `sys.Xdefaults`, for system-wide icons, or `.Xdefaults`, for your own personal icons.

- A text label.
- A graphic image.

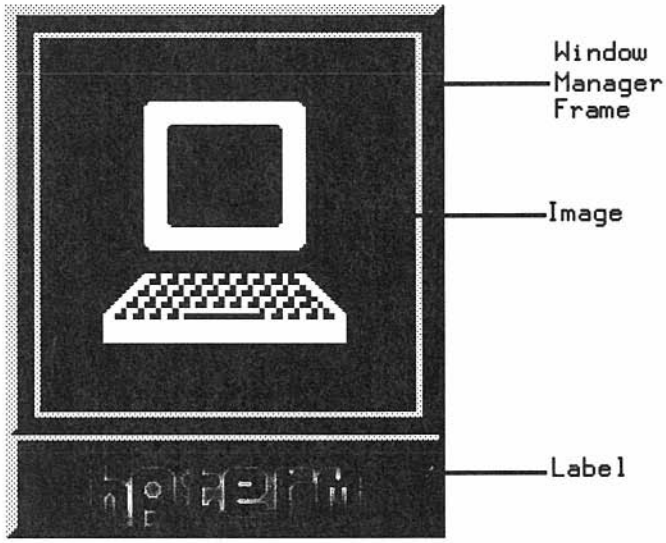


Figure 6-3. An Icon Has Two Parts.

The Label

An icon **label** is the text beneath an icon image. A label is usually supplied by the client (via the **WM_ICON_NAME** window property), but some clients, for example **hpterm** and **xclock**, provide a command-line option enabling you to write in your own label.

Icon labels are truncated on the right to the width of the icon image, so if you use small images, don't get too windy with your labels.

The Image

An icon image (a bitmap) is the actual graphic illustration of the icon. An image can come from any one of the following three sources:

client — A client can use the **WM_HINTS** window property to specify either an icon window or a bitmap for the window manager to use as the icon image.

user	You, the user, can specify an icon image using the `Hpwm*iconImage` or `Hpwm*default*iconImage` resources.
default	The window manager will use its own built-in default icon image if an image is not specified elsewhere.

The window manager uses the following order of precedence in choosing an icon image:

1. A specific user-supplied icon image resource.
2. A client-supplied icon window.
3. A client-specified icon image.
4. A default icon image.

Manipulating Icons

You manipulate icons similar to the way you manipulate windows, by positioning the pointer on the icon and clicking, double-clicking, or dragging a mouse button (depending on what you want to happen). You can also use icons in situations where you want to start several processes when you start X11, but don't want to clutter your screen with windows you won't immediately use; simply start the processes as icons.

Operating on Icons

The following table lists the operations you can perform on icons:

Table 6-6. You Can Manipulate Icons in These Ways.

To do this ...	Position the pointer on the icon and ...	What this does is ...
Turn an icon into a window.	Double-click the select button.	Restores the window to its former size and location.
Move an icon around on the root window.	Drag the select button.	Moves a wire frame with the pointer showing where the icon will be moved.
Move an icon to the top of the window stack.	Click the select button.	Moves a partially concealed icon to the front of the root window.
Select an icon and display its system menu.	Click the select button, then press Left Shift Esc.	Gives an icon keyboard focus and displays the icon's system menu. The system menu for an icon is exactly like the system menu of its associated window. No window is active while the icon has the keyboard focus.

Starting Clients as Icons

You can start clients as icons when you start X11. This gives you the benefit of having the client only a double-click away, while not cluttering your display with windows you're not using.

Some clients have iconify options, like `hpterm`'s `-iconic` option. As you start the client from a command line in your `.x11start` or `.hpwmrc` file, adding the iconify option to the line enables you to start the client but to display it initially as an icon. Later, when you're ready to use the client, you double-click on the icon and you're ready to go.

Controlling Icon Placement

By default, the window manager places icons in the lower left corner of the root window. Successive icons are placed in a row proceeding toward the right. Icons are prevented from overlapping. An icon will be placed in the position it last occupied if no icon is already there. If that place is taken, the icon will be placed at the next free location.

The following three resources enable you to control the placement of icons:

Table 6-7. Controlling Icon Placement with Window Manager Resources.

To specify this ...	Use this resource ...	The default value is ...
A placement scheme for icons.	`iconPlacement`	left bottom
The distance between screen edge and icons.	`iconPlacementMargin`	the default space between icons
Automatic icon placement by the window manager.	`iconAutoPlace`	True

Changing Screen Placement

You can place icons or you can have the window manager do it for you. The window manager will place icons automatically, based on the placement scheme you specify with the `iconPlacement` resource, if you give `iconAutoPlace` a value of "True." If you would rather determine icon placement without help from the window manager, give `iconAutoPlace` a value of "False."

The following table lists the icon placement schemes available to you:

Table 6-8. Schemes for Automatic Placement of Icons.

If you want this icon placement ...	Choose this scheme ...
From left to right across the top of the screen.	left top
From right to left across the top of the screen.	right top
From left to right across the bottom of the screen.	left bottom
From right to left across the bottom of the screen.	right bottom
From bottom to top along the left of the screen.	bottom left
From bottom to top along the right of the screen.	bottom right
From top to bottom along the left of the screen.	top left
From top to bottom along the right of the screen.	top right

The Syntax for Icon Placement Resources

The resources that place icons share a common syntax:

`Hpwm*`*resource value*

For example, if you want automatic placement of icons starting at the top of the screen and proceeding down the right side, you would have the following lines in your `.Xdefaults` file:

```
Hpwm*iconPlacement: top right     Specifies the placement scheme.
Hpwm*iconAutoPlace: True          Specifies automatic placement.
```

Controlling Icon Appearance and Behavior

The HP Window Manager offers you a number of resources to control the specific appearance and behavior of icons. Among these are resources that enable you to select icon decoration, control icon size, and create new icon pixmaps.

Selecting Icon Decoration

Using the `iconDecoration` resource, you can select exactly what parts of an icon you want to display:

Table 6-9. The Values That Control the Appearance of Icons.

If you want an icon that looks like this ...	Use this value ...
Just the label.	label
Just the image.	image
Both label and image.	label image
The label of an active icon isn't truncated.	label activelabel

Sizing Icons

Each icon image has a maximum and minimum size. The HP Window Manager has both default sizes as well as maximum and minimum allowable sizes.

Table 6-10. The Maximum and Minimum Sizes for Icon Images.

	Maximum Size	Minimum Size
Default	64×64 pixels	32×32 pixels
Allowable	128×128 pixels	16×16 pixels

If you plan to do a lot of work with icons, remember to keep your images within the maximum and minimum limits. How the window manager treats an icon depends on the size of the image in relation to the maximum and minimum sizes.

Table 6-11. Icon Size Affects Icon Treatment.

If an icon image is ...	The window manager will ...
Smaller than the minimum size.	Act as if you specified no image.
Within maximum and minimum limits.	Center the image within the maximum area.
Larger than the maximum size.	Clip the right side and bottom of the image to fit the maximum size.

You can use the following two resources to control icon image size:

Table 6-12. Controlling Icon Image Size with Window Manager Resources.

To specify this ...	Use this resource ...
Maximum size of an icon image.	`iconImageMaximum`
Minimum size of an icon image.	`iconImageMinimum`

If you figure icon size based on how much screen "real estate" you can afford to devote to icon space, bear in mind that the overall width of an icon is the image width *plus* border padding and the image height is the image height *plus* border padding.

Using Custom Pixmaps

When you iconify a client, either the client supplies its own icon image, the window manager supplies a default image, or you supply an image of your own.

Some icon images you will obtain as "ready-made" bitmaps. At other times, you may want to use the `bitmap` client, discussed in chapter 5 to create one of your own. In either case, to use your bitmap, you only need to tell the window manager where the bitmap is located.

To tell the window manager to use a particular bitmap for an icon image, you use the `iconImage` resource. The value that follows this resource is the path to the bitmap file you want to use. Note that, if specified, this resource overrides any client-specified image.

You also have the ability, using the **bitmapDirectory** resource, to direct the window manager to search a specified directory for bitmaps. The **bitmapDirectory** resource causes the window manager to search the specified directory whenever a bitmap is named with no complete path. The default value for **bitmapDirectory** is **/usr/include/X11/bitmaps**.

The Syntax for Resources that Control Icon Appearance

The resources that control icons appearance have the following syntax:

Hpwm**resource*: *value*

For example, you could use **bitmapDirectory** to search a **bitmap** subdirectory in your home directory for custom bitmaps by inserting the following line in your **.Xdefaults** file:

Hpwm*bitmapDirectory: /users/*yourusername***/bitmap**

Additionally, the **iconImage** resource has two other syntaxes. Which of the three you use depends on which of the following statements is true:

Table 6-13. The 'iconImage' Resource Has Three Syntaxes.

If this is true ...	Use this syntax ...
You want to use the image for *all* clients for which you don't otherwise specify an image. All these clients will have the *same* image.	**Hpwm*iconImage:** *path/bitmap*
You want to use the image *only* for a specific class of clients.	**Hpwm*** *clientclass*.**iconImage:** *path/bitmap*
You want to use the image as the default image whenever the client class isn't known.	**Hpwm*defaults*iconImage:** *path/bitmap*

For example, if you want to use your own **happyface** bitmap for **hpterm** windows and see a complete label whenever any icon is active, you would have the following lines in your **.Xdefaults** file:

```
Hpwm*Hpterm*iconImage: /users/yourusername/Bitmaps/face.bits
Hpwm*iconDecoration: label activelabel
```

Coloring Icons by Client Class

As it does for window frames, the HP Window Manager supplies a number of resources that enable you to specify the colors of icon elements.

Coloring Icon Elements Individually

The following table lists icon image elements and the resources that control their color.

Table 6-14. Coloring Icons by Client Class with Window Manager Resources.

To color this ...	Use this resource ...
Icon image background.	`iconImageBackground`
Left and upper bevel of icon image.	`iconImageTopShadowColor`
Right and lower bevel of icon image.	`iconImageBottomShadowColor`
Icon image foreground.	`iconImageForeground`

To color these elements individually, you must include another icon resource, `iconColors`, in your `.Xdefaults` file. If the value of `iconColors` is "True," the above colors are used to color the icon image. If the value for `iconColors` is "False," the default colors are used to color the icon image. The default value for `iconColors` is "False."

If you do not choose to color an element of an icon image, the window manager will use default values. It gets these values from either of the two following lines:

`Hpwm*`*resource*`:` *color* *Colors every instance of an element.*

`Hpwm*icon*`*resource*`:` *color* *Colors only this element of icons.*

You can find these lines in the `.Xdefaults` and `sys.Xdefaults` files.

When making changes, don't confuse an *element* (foreground, background, topShadowColor) with a *resource* (iconImageForeground, iconImageBackground, iconImageTopShadowColor).

Coloring Icon Elements Automatically

Another resource enables you to color icon elements automatically, thus avoiding the need of specifying the colors individually.

The `makeIconColors` resource uses the `iconImageBackground` color to generate colors for the other icon image elements, giving the icon a 3-D look.

Exactly which elements you color with `makeIconColors` depends on which of three values you give the resource. The resource values and their effect on an icon image elements are as follows:

Table 6-15. The Values to Use for Automatically Coloring Frame Elements.

To color these elements ...	Use this value ...
All bevel elements and the foreground (tile elements for top and bottom shadows are set to foreground).	`all`
All bevel elements.	`shadow`
No elements automatically colored.	`none`

Changing the Tile of Icon Images

The HP Window Manager also has resources that enable you to tile the bevels of icon images.

Table 6-16. Tiling Icon Images with Window Manager Resources.

To tile this ...	Use this resource ...
Right and lower bevels of an icon image.	`iconImageBottomShadowTile`
Left and upper bevels of an icon image.	`iconImageTopShadowTile`

Default values for these resources are the bottom and top shadow tiles, respectively as specified in `.Xdefaults` or `sys.Xdefaults` by either an `Hpwm*`*iconImageTile* or an `Hpwm*icon*tile` entry.

The Syntax for Icon Coloring Resources

The resources that color icons can have any of three different syntaxes.

The first syntax is the most specific. You can use it to specify not only a specific resource and color, but a specific client class to which to apply the resource. The colors you specify with this resource take precedence over any other specification for this resource.

`Hpwm*`*clientclass*`*`*resource*: *color*

The second syntax enables you to specify a resource and value generally across any and all clients. An example of proper use would be to ensure that all your icon backgrounds were the same color, a good thing for consistency.

`Hpwm*`*resource color*

The third syntax is a default syntax. Using it you can specify that any client that is of unknown class will have the color you select.

`Hpwm*default*`*resource color*

Managing Window Manager Menus

Menus offer an easy way to get the system to do something for you. Most new users, while the ideas of "entering commands" and "arguments" to the "operating system" is totally foreign to them, can readily appreciate the concept of choosing a selection from a menu.

Default Menus

The HP Window Manager comes with two default menus:

- The System Menu.
- The Root Menu.

The default system menu is specified in `/usr/lib/X11/system.hpwmrc` and in your home directory in `.hpwmrc` by the following lines:

```
Menu DefaultSystemMenu          Function type and name for system menu.
{
 Restore     f.normalize        Normalizes icon or maximized window.
 Move        f.move             Moves window around screen.
 Size        f.resize           Changes window size.
 Minimize    f.minimize         Changes window into icon.
 Maximize    f.maximize         Enlarges window to cover screen.
 Lower       f.lower            Lowers window to bottom of stack.
 Close       f.kill             Closes window by killing its process.
}
```

The default root menu is specified in the same files by the following lines:

```
Menu DefaultRootMenu                    Function type and name for
                                        root menu.
{
 "Root Menu"     f.title                Gives root menu's title.
 "New Window"    f.exec "hpterm &"      Starts hpterm client.
 "Start Clock"   f.exec "xclock &"      Starts xclock client.
 "Start Load"    f.exec "xload &"       Starts xload client.
 "Shuffle Up"    f.circle_up            Lifts window to top of stack.
 "Shuffle Down"  f.circle_down          Lowers window to bottom of
                                        stack.
 "Refresh"       f.refresh              Repaints root window.
  no-label       f.separator            Draws line across menu.
 "Restart"       f.restart              Restarts window manager.
}
```

By default, the system menu displays when you do the following operations:

- Click the select button on a window frame's system menu button.

- Click the menu button anywhere on a window frame.
- Press Shift Esc with the pointer in a window.

By default, the root menu displays when you click the menu button on the root window.

You can modify either menu to suit the specific needs of your application; however, for the sake of the consistency of window operation, it's usually better to modify the root menu and keep the system menu the same.

Modifying Menu Selections and Their Functions

All window manager menus, regardless of the mechanism that calls them to the screen, have the same syntax.

Menu Syntax

```
Menu MenuName
{
  [ selection1 function [arguments]
    selection2 function [arguments]
    selection3 function [arguments]
    selection* function [arguments] ]
}
```

Each line identifies a selection name followed by the function to be done if that selection is chosen. The order of the selections is the order of their appearance when you display the menu. A selection name may be either a character string or a bitmap.

Modifying Selections

Any character string containing a space must be enclosed in double quotes (“”); single-word strings don't have to be enclosed, but it's probably a good idea for the sake of consistency. An alternate method of dealing with two-word selection names is to use an underbar (_) in place of the space.

You can create a bitmap with the `bitmap` client and use it as a selection name. The syntax for doing this is as follows:

@*/path/bitmapfile function* [*argument*]

Note The at-sign (@) in the above line. The at-sign tells the window manager that what follows is the path to a bitmap file.

Modifying Functions

Each function operates in one or more of the following contexts:

root	Operates the function when the root window is selected.
window	Operates the function when a client window is selected. Some functions operate only when the window is in its normalized state (f.maximize), or its maximized or iconified state (f.normalize).
icon	Operates the function when an icon is selected.

Additionally each function is operated by one or more of the following devices:

- Button.
- Key.
- Menu.

Any selection that uses an invalid context, an invalid function, or a function that doesn't apply to the current context is grayed out. This is the case with the "Restore" selection of a terminal window's system menu or the "Minimize" selection of an icon's system menu.

valid| The following table lists the valid functions for the HP Window Manager.

Table 6-17. Valid Window Manager Functions.

Functions		Contexts			Devices		
Name	**Description**	**Root**	**Icon**	**Window**	**Button**	**Key**	**Menu**
f.beep	Causes a beep to sound.	√	√	√	√	√	√
f.circle_ down	Puts window on bottom of stack.	√	√	√	√	√	√
f.circle_ up	Puts window on top of stack.	√	√	√	√	√	√
f.exec	Uses `/bin/sh` to execute a command.	√	√	√	√	√	√
f.focus_ color	Sets colormap focus when colormap focus policy is explicit.	√	√	√	√	√	√
f.focus_ key	Sets keyboard input focus when keyboard focus policy is explicit.	√	√	√	√	√	√
f.kill	Terminates a client's connection to server.		√	√			√

Table 6-17a. Valid Window Manager Functions (continued).

Functions		Contexts			Devices		
Name	Description	Root	Icon	Window	Button	Key	Menu
f.lower	Lowers a window to bottom of stack.		√	√	√	√	√
f.maximize	Enlarges a window to its maximum size.		√	√	√	√	√
f.menu	Associates a menu with a selection or binding.	√	√	√	√	√	√
f.minimize	Changes a window into an icon.			√	√	√	√
f.move	Enables the interactive moving of a window.		√	√	√	√	√
f.next_ cmap	Installs the next colormap in the window with the colormap focus.	√	√	√	√	√	√
f.next_ key	Sets keyboard focus policy to the next window/icon in the stack.	√	√	√	√	√	√
f.nop	Does no function.	√	√	√	√	√	√
f.normalize	Displays a window in normal size.		√	√	√	√	√
f.prev_ cmap	Installs the previous color map in the window with the colormap focus.	√	√	√	√	√	√

Table 6-17b. Valid Window Manager Functions (continued).

Functions		Contexts			Devices		
Name	Description	Root	Icon	Window	Button	Key	Menu
f.prev_ key	Sets the keyboard input focus to the next window/icon in the stack.	√	√	√	√	√	√
f.post_ smenu	Posts the system menu.	√	√	√	√	√	
f.quit_ hpwm	Terminates HP Window Manager, but not X.	√					√
f.raise	Lifts a window to the top of the window stack.		√	√	√	√	√
f.raise_ lower	Raises a partially concealed window; lowers an unconcealed window.		√	√	√	√	√
f.refresh	Redraws all windows.	√	√	√	√	√	√
f.refresh_ win	Redraws a client window.			√	√	√	√
f.resize	Enables you to interactively resize a window.			√	√	√	√
f.restart	Restarts the HP Window Manager.	√			√	√	
f.separator	Draws a line between menu selections.	√	√	√			√
f.title	Inserts a title in a menu selection.	√	√	√			√

Changing the Menu Associated with the System Menu Button

The `f.title` function enables you to change the title of the menu that displays when you press the select button on the system menu button. The `systemMenu`

resource enables you to change the *menu* that displays when you press the select button on the system menu button.

This gives you the ability to display a menu of your choice from the system menu button without having to extensively remodel the system menu to do it. All you need do is make a new menu, then use the **systemMenu** resource to associate this new menu with the system menu button.

The **systemMenu** resource has three syntaxes. Which one you use depends on your situation.

The first syntax specifies the menu for all classes of clients:

Hpwm*systemMenu: *MenuName*

For example, if you want to associate a special **CADCAMMenu** menu with the system menu button, you would add the following line to your **.Xdefaults** file:

```
Hpwm*systemMenu: CADCAMMenu
```

The second syntax specifies the menu for a specific class of clients:

Hpwm* *clientclass* **.systemMenu:** *MenuName*

For example, you may want to associate a particular **EditorMenu** of your own creation with **hpterm** windows:

```
Hpwm*HPterm.systemMenu:  EditorMenu
```

The third syntax is for specifying a menu for any client whose class is unknown:

Hpwm*defaults*systemMenu: *MenuName*

Making New Menus

You have the option of modifying the system and root menus, but you also have another option: You can create a completely new menu, calling it to the screen either by a button press, by a key press, or by selecting it from an existing menu.

To create a completely new menu, use the above menu syntax as a model to do the following:

1. Fill in a menu name.
2. Make up selection names.
3. Give each selection a function to perform from the "Table of Menu Functions."

For example, the following menu is named "Graphics Projects." The menu selections are all bitmaps symbolizing different graphics projects. The bitmaps are kept in this user's home directory **/users/dub/bits**. When the user, Dub, selects a symbol, the graphics program starts and opens the appropriate graphics file.

```
Menu "Graphics Projects"
{
 @/users/dub/bits/fusel.bits    f.exec "cad /spacestar/fusel.e12
 @/users/dub/bits/lwing.bits    f.exec "cad /spacestar/lwing.s05
 @/users/dub/bits/rwing.bits    f.exec "cad /spacestar/rwing.s04
 @/users/dub/bits/nose.bits     f.exec "cad /spacestar/nose.e17
}
```

To display a new menu with a button or key, press follow these steps:

1. Choose the button or key that you want to use.
2. Choose the action on the button or key that will cause the menu to display.
3. Use the **f.menu** function with the menu name as an argument to bind the menu to the button or key.

For more information on button and keyboard bindings, including examples, see the next two sections, "Using the Mouse" and "Using the Keyboard."

Using the Mouse

The mouse offers a quick way to make things happen in your window environment without having to undergo the time-consuming process of typing commands on the keyboard (and retyping misspelled commands). The window manager recognizes the following button operations:

Press Holding down a mouse button.

Click Pressing and releasing a mouse button.

Double-click Pressing and releasing a mouse button twice in rapid succession.

Drag Pressing a mouse button and moving the pointer (and mouse device).

You associate a button operation with a window management function using a **button binding.** A button binding is a command line you put in the `system.hpwmrc` or `.hpwmrc` file that associates a button operation with a window manager function.

Default Button Bindings

The HP Window Manager comes with the following default button bindings.

Table 6-18. HP Window Manager Default Button Bindings.

To do this ...	Use this action ...
Display system menu without using the system menu button.	Press the menu button with the pointer anywhere on a window frame).
Display the root menu.	Press the menu button with the pointer anywhere on the root window.

These bindings are specified by the following lines in **system.hpwmrc** and **.hpwmrc**.

```
Buttons DefaultButtonBindings
{
 <Btn3Down>  root  f.menu  DefaultRootMenu
 <Btn3Down>  frame f.post_smenu
}
```

Modifying Button Bindings and Their Functions

You can modify the button bindings section of your **.hpwmrc** file to suit your individual needs.

Button Binding Syntax

The syntax for button bindings is as follows:

```
Buttons ButtonBindingSetName
{
    button   context|context   function   argument
    button   context|context   function   argument
    button   context|context   function   argument
}
```

Each line identifies a certain button and operation, followed by the context in which the button operation is valid, followed by the function to be done. The following button binding contexts are recognized by the window manager:

root	Operates the function when the button is activated in the root window.
window	Operates the function when the button is activated in a client window.
frame	Operates the function when the button is activated on a window frame.
icon	Operates the function when the button is activated on an icon.
title	Operates the function when the button is activated on a title bar.

Modifying Button Bindings

To modify the default button bindings, you need to edit either `system.hpwmrc` (to make system-wide changes) or `.hpwmrc` (to make changes to your local environment). The easiest way to modify button bindings is to change the default bindings or to insert extra lines in the "DefaultButtonBindings" section.

For example, Dub, the user who created his own "Graphics Project" menu in the previous section, may want to display the menu when he presses the **Extend char**–menu button sequence with the pointer in the root window. He would only need to insert one line in his `.hpwmrc` file to make this happen. The "DefaultButtonBindings" section of his `.hpwmrc` file would look like the following.

```
Buttons DefaultButtonBindings
{
 <Btn3Down>       root  f.menu  DefaultRootMenu
 <Btn3Down>       frame f.post_smenu
 Meta<Btn3Down>   root  f.menu  "Graphics Project"
}
```

Making a New Button Binding Set

Perhaps inserting a new button binding into the "DefaultButtonBindings" set is not enough. Perhaps you need to make a complete new set of button bindings. To do this, use the "DefaultButtonBindings" section of your `.hpwmrc` as a model. After you have created the new button binding set, you need to use the `buttonBindings` resource to tell the window manager about it. You do this by adding a line to your `.Xdefaults` file. The syntax of the line is as follows:

`Hpwm*buttonBindings:` *NewButtonBindingSet*

This line directs the window manager to use "NewButtonBindingSet" as the source of its button binding information. The button bindings are assumed to exist in the file named by the `Hpwm*configFile:` resource, the default being `.hpwmrc`.

For example, suppose Dub, our graphics user, wants to specify a completely new button binding set instead of inserting a line in the existing

"DefaultButtonBindings" set. He needs to create a new button binding set, such as the following, modeled after the default set:

```
Buttons GraphicsButtonBindings
{
 <Btn2Down>  root  f.menu  "Graphics Project"
}
```

In his `.Xdefaults` file, Dub would then insert the following line:

```
Hpwm*buttonBindings: "Graphics Project"
```

To display his graphics menu, Dub needs only to press the alternate button when the pointer is on the root window.

Modifying Button Click Timing

The HP Window Manager has another resource for controlling button behavior. This resource, `doubleClickTime`, sets the maximum time (in milliseconds) that can elapse between button clicks before a double-click becomes just "two clicks in a row." In other words, if two clicks occur in less than the maximum time, they are assumed to be a double-click; if two clicks occur in a time greater than the maximum time, they are assumed to be two single clicks.

Using the Keyboard

Similar to mouse button bindings, you can bind (associate) window manager functions to "special" keys on the keyboard using **keyboard bindings.** The window manager recognizes the following special keys:

- Shift.
- Escape.
- Meta (Extend Char).
- Tab.

Default Keyboard Bindings

The HP Window Manager comes with the following default keyboard bindings.

Table 6-19. HP Window Manager Default Keyboard Bindings.

To do this ...	Press these keys ...
Display system menu without using the system menu button.	Left Shift Esc (with pointer anywhere in the active window).
Set keyboard focus to the next window in the stack.	Extend char Tab (with pointer anywhere in the active window and explicit keyboard focus).

Keyboard bindings are specified by the following lines in `system.hpwmrc` and `.hpwmrc`.

```
Keys DefaultKeyBindings
{
 Shift<Key>Escape    icon|window    f.post_smenu    DefaultSystemMenu
 Meta<Key>Tab        window         f.next_key
}
```

The first line specifies the function type (`Keys`) and the name of the keyboard binding set (`DefaultKeyBindings`). The following two lines specify the actual key bindings. The first line binds the Shift Esc key press sequence to the function that displays the system menu. The second line binds the Extend char and Tab key press sequence to the function that selects the next window in the stack for keyboard focus.

Modifying Button Bindings and Their Functions

You can modify the keyboard bindings section of your `.hpwmrc` file if your situation requires it.

Keyboard Binding Syntax

The syntax for keyboard bindings is as follows:

```
Keys KeyBindingSetName
{
    key   context|context   function   argument
    key   context|context   function   argument
    key   context|context   function   argument
}
```

Each line identifies a unique key press sequence, followed by the context in which that sequence is valid, followed by the function to be done. The following keyboard binding contexts are recognized by the window manager:

root	Operates the function when the button is activated in the root window.
window	Operates the function when the button is activated in a client window.
frame	Operates the function when the button is activated on a window frame.
icon	Operates the function when the button is activated on an icon.
title	Operates the function when the button is activated on a title bar.

Modifying Keyboard Bindings

To modify the default keyboard bindings, you need to edit either `system.hpwmrc` (to make system-wide changes) or `.hpwmrc` (to make changes to your local environment). The easiest way is to change the default bindings or to insert extra lines in the "DefaultKeyBindings" section.

For example, suppose Dub, the user who created his own "Graphics Project" menu, kept pressing the Left Shift Esc sequence and accidentally displaying the system menu. He might decide that he is better off to disable that particular keyboard binding. To do so, he would need to delete (or comment out) the proper line in his `.hpwmrc` file. The "DefaultKeyBindings" section of his `.hpwmrc` file would then look like the following.

```
Keys DefaultKeyBindings
{
# Shift<Key>Escape  icon|window  f.post_smenu DefaultSystemMenu
 Meta<Key>Tab       window       f.next_key
}
```

Dub has chosen simply to comment out the line by placing a hash mark (#) in the left margin of the line.

Making a New Keyboard Binding Set

With keyboard bindings, as with button bindings, you have the option of creating a whole new binding set. To do so, you can use the "DefaultKeyBindings" section of your `.hpwmrc` as a model. After you have created the new keyboard binding set, use the **keyBindings** resource to explain your modification to the window manager. You do this by adding a line to your `.Xdefaults` file. The syntax of the line is as follows:

`Hpwm*keyBindings:` *NewKeyboardBindingSet*

This line directs the window manager to get its keyboard binding information from the "NewKeyboardBindingSet" section of the `.hpwmrc` file. You can have the window manager look in any file if you specify the path and file name with the `Hpwm*configFile:` resource in your `.Xdefaults`.

Using Windows without Frames

In some cases, you might feel your "screen real estate" is too expensive, and you may not want to take up "valuable space" with window frames. For example, do you really need functional buttons and a resizable frame around a clock that just sits in the corner of your screen? You could switch to the `uwm` window manager, but the HP Window Manager has two resources for just such situations. The `clientDecoration` resource enables you to choose just how much or how little "decoration" you want to put around each client. The `transientDecoration` resource enables you to choose just how much or how little decoration you want to put around each transient window. A **transient window** is a relatively short-lived window, for example, a dialog box. You can still use any decoration you remove by binding its functions either to buttons or to key presses as explained in the above sections.

Adding or Removing Elements

You specify the `clientDecoration` and `transientDecoration` resources as a list of the frame elements. If the first element in the list is preceded by a plus (+) sign, the window manager starts with no frame and assumes that the list contains those elements you want added. If the list begins with a minus (−) sign, the window manager starts with a complete frame and assumes that the list contains elements you want removed from the frame.

The list of valid window frame elements is as follows:

Table 6-20. Valid HP Window Manager Window Frame Elements.

Frame Element	Description
All	Includes all window frame elements.
Maximize	Maximize button only.
Minimize	Minimize button only.
None	Includes no window frame elements.
Resize	Resize border (outermost portion of frame).
System	System menu.
Title	Title bar only.

The Syntax for the 'clientDecoration' and 'transientDecoration' Resources

The `clientDecoration` resource has three syntaxes. Which you use depends on your situation.

The first syntax is for specifying the addition or removal of elements from all classes of clients:

```
                                  [ { ±All  }  ]
                                  [ { ±None } ]
                                  [ ±Maximize ]
Hpwm*clientDecoration:            [ ±Minimize ]
                                  [ ±Resize   ]
                                  [ ±System   ]
                                  [ ±Title    ]
```

For example, you could remove the maximize button from all windows by adding the following line in your `.Xdefaults` file:

```
Hpwm*clientDecoration: -Maximize
```

The second syntax is for specifying the addition or removal of elements from specific classes of clients:

```
Hpwm*clientclass.clientDecoration: [ { ±All } ]
                                   [ { ±None} ]
                                   [ ±Maximize ]
                                   [ ±Minimize ]
                                   [ ±Resize   ]
                                   [ ±System   ]
                                   [ ±Title    ]
```

For example, you may want to remove just the resize border and maximize button from all clocks you display on your screen:

```
Hpwm*XClock.clientDecoration: -Resize -Maximize
```

The third syntax is for specifying the addition or removal of elements from any client with an unknown class:

```
Hpwm*defaults*clientDecoration: [ { ±All } ]
                                [ { ±None} ]
                                [ ±Maximize ]
                                [ ±Minimize ]
                                [ ±Resize   ]
                                [ ±System   ]
                                [ ±Title    ]
```

The `transientDecoration` resource has the following syntax:

```
Hpwm*transientDecoration: [ { ±All } ]
                          [ { ±None} ]
                          [ ±Maximize ]
                          [ ±Minimize ]
                          [ ±Resize   ]
                          [ ±System   ]
                          [ ±Title    ]
```

For example, you could remove the tile bar from all transient windows by adding the following line in your `.Xdefaults` file:

```
Hpwm*transientDecoration: -Title
```

Controlling Window Size and Placement

The HP Window Manager has several resources that allow you to refine your control of the size and placement of windows.

Refining Control with Window Manager Resources

The following table lists window manager resources enabling you to refine your control over the size and placement of windows.

Table 6-21. Refining Your Control with Window Manager Resources.

To control this ...	Use this resource ...	The default is ...
Initial placement of new windows on the screen.	`interactivePlacement`	False
The ability to enlarge a window beyond the size specified in `maximumClientSize`.	`limitResize`	False
The maximum size of a client window as set by either user or client.	`maximumMaximumSize`	2×screen
The sensitivity of dragging operations.	`moveThreshold`	4 pixels
Exact positioning of window and window frame.	`positionIsFrame`	True
Clipping of new windows by screen edges.	`positionOnScreen`	True
The width of the resize border of the window frame.	`resizeBorderWidth`	10 pixels
Displaying the resize cursors when the pointer is in the resize border.	`resizeCursors`	True
The maximum size of a maximized client.	`maximumClientSize`	screen size

The `interactivePlacement` resource has the following values:

True — The pointer changes shape (to an upper left corner bracket) before a new window displays, so you can choose a position for the window.

False — The pointer doesn't change shape. A new window displays according to the placement values specified in the X configuration files.

The `limitResize` resource has the following values:

True — A window cannot be resized to greater than the maximum size specified by the **maximumClientSize** resource or the **WM_NORMAL_HINTS** window property.

False — A window can be resized to any size.

The value of the **maximumMaximumSize** resource is the width×height of the screen being used. The dimensions are given in pixels. For example, for an SRX display, **maximumMaximumSize** would have a value of 1280×1024.

The value of the **moveThreshold** resource is the number of pixels that the pointer must be moved with a button pressed before a move operation is initiated. You can use this resource to prevent window or icon movement when you unintentionally move the pointer during a click or double-click.

The **positionIsFrame** resource has the following values:

True — The position information (from **WM_NORMAL_HINTS** and configuration files) refers to the position of the window frame.

False — The position information refers to the position of the window itself.

The **positionOnScreen** resource has the following values:

True — If possible, a window is placed so that it is not clipped. If not possible, a window is placed so that at least the upper left corner of the window is on the screen.

False — A window is placed at the requested position even if it is totally off the screen.

The value of the **resizeBorderWidth** resource is the width of the resize border, the outermost portion of the window frame. The width is measured in pixels.

The **resizeCursors** resource has the following values:

True — The appropriate resize cursor displays when the pointer enters a resize border area of the window frame.

False — The resize cursors are not displayed.

The value of the **maximumClientSize** resource is the width×height (in pixels) of the maximum size of a maximized client. If this resource isn't specified, the

maximum size is taken from the **WM-NORMAL_HINTS** window property, or the default size (the size of the screen) is used.

The Syntax for Size and Position Refinement Resources

The resources that refine your control over the size and placement of windows have the following syntax:

Hpwm* *resource*: *value*

For example, you could choose to place each new window on the screen interactively by adding the following line in your **.Xdefaults** file:

```
Hpwm*interactivePlacement: True
```

In addition to this syntax, the **maximumClientSize** resource has two more syntaxes.

Use this syntax is for specifying the maximum client size for specific classes of clients:

Hpwm* *clientclass*.**maximumClientSize**: *width*×*height*

For example, you might decide that **xload** clients should be maximized to no more than an eighth of the size of your 1024×768 display.

```
Hpwm*XLoad.maximumClientSize: 128×96
```

Use this syntax is for specifying the maximum client size for any client with an unknown class.

Hpwm*defaults*maximumClientSize: *width*×*height*

Controlling Resources with Focus Policies

The HP Window Manager has three separate focus policies that you can use to control the arbitration of resources among clients. The focus policies determine what happens when a window becomes the active window. The active window is the window that has the focus of the keyboard, the colormap, and any extended input devices. When a window is active, the following are true:

- What you type appears in that window.
- The color of the window frame changes to indicate the active focus.
- Input from extended input devices goes to that window.

Each focus policy is controlled by a specific focus policy resource. The focus policy resources are as follows:

Table 6-22. Controlling Focus Policies with Window Manager Resources.

To control this ...	Use this resource ...	The default value is ...
Which client window has the colormap focus.	`colormapFocusPolicy`	keyboard
Which client window has the keyboard and mouse focus.	`keyboardFocusPolicy`	explicit

Valid Focus Policies

The following focus policies are valid for the `colormapFocusPolicy` resource:

keyboard — The window manager tracks keyboard input and installs a client's colormap when the client window gets the keyboard input focus.

pointer — The window manager tracks the pointer and installs a client's colormap when the pointer moves into the client window or the window frame around the client.

explicit — The window manager tracks a specific focus-selection operation and installs a client's color map when the focus-selection operation is done in the client window.

The following focus policies are valid for the **keyboardFocusPolicy** resource:

pointer	The window manager tracks the pointer and sets the keyboard focus to a client window when the pointer moves into that window or the window frame around the client.
explicit	The window manager tracks a specific focus-selection operation and sets the keyboard focus to a client window when the focus-selection operation is done in that client window.

When the keyboard focus policy is explicit, you can use the **passSelectButton** resource to specify the consequence of the focus-selection operation. If you give **passSelectButton** a value of "True" (the default value), the focus-selection operation is passed to the client or used by the window manager to perform some action. If you give **passSelectButton** a value of "False," the focus-selection operation will be used only to select the focus and will not be passed.

The Syntax of Focus Policy Resources

All four focus policy resources have the following syntax:

```
Hpwm*focusPolicyResource: policy
```

For example, you could change the keyboard focus policy so that moving the pointer into a window moved the focus there by adding the following line in your **.Xdefaults** file:

```
Hpwm*keyboardFocusPolicy: pointer
```

Matting Clients

If you have a color system, you might find it useful to decorate windows based on client class. For example, you may wish to color code your **hpterm** windows so you can easily tell them apart from your **xterm** windows.

You can accomplish this differentiation by using a **matte** and the window manager's matte resources to further frame your client windows. A matte is a 3-D border just inside the window between client area and window frame.

To enable a matte, define a positive matte width for windows of a specific class. Careful selection of matte elements will give a pleasing 3-D effect.

To define a matte width, use the **matteWidth** resource. The **matteWidth** resource defines the width of the matte or border between the client and the window frame. The width is given in pixels. For example, to specify a matte of 10 pixels around **hpterm** windows, you would include the following line in your **.Xdefaults** file:

```
Hpwm*HPterm.mattewidth:  10
```

Coloring Individual Matte Elements

The following table lists matte elements and the resources that control their color.

Table 6-23. Coloring Window Frames with Window Manager Resources.

To color this ...	Use this resource ...	The default value is ...
Matte background.	**matteBackground**	client background
Left and upper bevel of matte.	**matteTopShadowColor**	client top shadow color
Right and lower bevel of matte.	**matteBottomShadowColor**	client bottom shadow color
Matte foreground.	**matteForeground**	client foreground

Coloring Matte Elements Automatically

Additionally, the window manager has a resource that enables you to color matte elements automatically (without specifying the color of each element).

The **makeMatteColors** resource uses the background color of the matte background to generate colors for the other matte elements, giving the frame a 3-D look.

Exactly which elements you color with **makeMatteColors** depends on which of the following three values you give the resource:

Table 6-24. The Values to Use for Automatically Coloring Matte Elements.

To color these elements ...	Use this value ...
All bevel elements and the foreground (tile elements for top and bottom shadows are set to foreground).	**all**
All bevel elements.	**shadow**
No elements automatically colored.	**none**

Changing the Tile of Mattes

As with frame colors, the fewer the number of colors your display can produce, the more interest you will probably have in tiling mattes. Again, tiling provides you with a way to "mix" foreground and background colors into a third color. If you have a 2-color (monochrome) display, you could tile a window matte in shades of gray to achieve a pleasing 3-D look.

The HP Window Manager has the following resources to enable you to tile mattes.

Table 6-25. Tiling Mattes with Window Manager Resources.

To tile this ...	Use this resource ...	The default value is ...
Matte right and lower bevels.	**matteBottomShadowTile**	client bottom shadow color
Matte left and upper bevels.	**matteTopShadowTile**	client top shadow color

The following table lists the acceptable values for tile resources:

Table 6-26. The Values to Use for Tiling Window Frames.

To tile an element this color mix ...	Use this value ...
The foreground color.	foreground
The background color.	background
A mix of 25% foreground to 75% background.	25_ foreground
A mix of 50% foreground to 50% background.	50_ foreground
A mix of 75% foreground to 25% background.	75_ foreground
In horizontal lines alternating between the foreground and background color.	horizontal_ tile
In vertical lines alternating between the foreground and background color.	vertical_ tile
In diagonal lines slanting to the right alternating between the foreground and background color.	slant_ right
In diagonal lines slanting to the left alternating between the foreground and background color.	slant_ left

The Syntax for Matte Resources

Matte resources can have any of the following three syntaxes, depending on the situation:

Use the first syntax to create a matte for all clients regardless of class:

```
Hpwm*resource: value
```

For example, you could create a 10-pixel-wide yellow matte for every client window by adding the following lines in your `.Xdefaults` file:

```
Hpwm*matteWidth:        10
Hpwm*matteBackground:   Yellow
Hpwm*makeMatteColors:   all
```

Use the second syntax to specify the matte for specific classes of clients:

`Hpwm*`*clientclass.matteResource*`:` *value*

For example, as mentioned earlier, you could place a different matte around **hpterm** and **xterm** windows by including the following lines in your **.Xdefaults** file:

```
Hpwm*HPterm.matteWidth:          10
Hpwm*HPterm.matteBackground:     SkyBlue
Hpwm*HPterm.makeMatteColors:     all
Hpwm*Xterm.matteWidth:           10
Hpwm*Xterm.matteBackground:      Tan
Hpwm*Xterm.makeMatteColors:      all
```

The third syntax specifies the matte for any client with an unknown class.

`Hpwm*defaults*matteWidth:` *width*

What's Next

The next two chapters cover special custom-environment situations. Chapter 7 concerns special hardware-related configurations. Chapter 8 concerns the use of Starbase graphics and the printing of screen dumps.

7

Customizing Special X Environments

Some applications, perhaps yours is one of them, require customization beyond changing colors, choosing clients for your environment, and modifying window manager resources. This chapter describes the following "special" X Window System customizations:

- Using custom screen configurations.
- Using special input devices.
- Going mouseless.
- Customizing keyboard input.
- Creating a custom color database.
- Changing display preferences.
- Compiling `bdf` fonts into `snf` format.
- Using 'xrdb' to configure the server.
- Using national languages.

This chapter discusses the following X clients:

xmodmap	Modifies keyboard bindings to modifier keys.
xset	Sets user preferences for display behavior.
rgb	Creates a custom color database.
xfc	Compiles `bdf` format fonts into `snf` format fonts.
xrdb	Initializes the X server with resource specifications.

More information about these clients is in the reference section.

Using Custom Screen Configurations

The default screen configuration for X11 assumes the following about your system:

- You use display 0 (typically the console).
- You have only one physical display screen.
- Your screen uses only one set of pixel planes (the image planes).
- Your screen is at the address node specified by **/dev/crt**.

However, the configuration you actually have may differ from the default configuration. For example, you may be display 1. Or you may be display 0, but have "two heads" (screens). Or you may be screen 0 on display 0, but have two sets of planes, image and overlay. There are many possible combinations of non-default screen configurations.

If you use some configuration other than the default, you must create a custom screen configuration file for yourself.

The Default Screen Configuration File

The default screen configuration file is located in **usr/lib/X11** and is called **X0screens** (x zero screens). The "0" is an arbitrary number chosen to signify the default display.

X0screens is an ASCII file. It contains the path and name of the default screen, bitmap device **/dev/crt**.

Creating a Custom 'X*screens' File

To tell the server about a custom screen configuration, you need either to modify **X0screens** or to create an **X*screens** file. The * should be replaced by a display number signifying the new configuration.

Each **X*screens** file represents a custom screen configuration and can contain a maximum of four lines (excluding comments and blank lines).

As an alternative to specifying multiple **X*screens** files, you can make one **X*screens** file for your display. This file will contain the device information for all the configurations. But only one configuration will be in use at a time;

the other configurations will be commented out. To switch configurations, instead of choosing a different `X*screens`, you edit your single configuration file, uncommenting only the configuration to which you want to switch.

Choosing a Screen Mode

Although each `X*screens` file contains lines listing a screen device, the exact syntax of the lines depends on the screen mode you choose. Depending on your system's hardware, you may choose from four screen modes:

image mode	The default screen mode using multiple image planes for a single screen. The number of planes determines the variety of colors available to the screen.
overlay mode	An alternate screen mode using overlay planes for a single screen. Overlay planes provide an alternate (auxiliary) set of planes to the standard image planes. Typically, overlay planes are used in conjunction with image planes in either stacked mode or combined mode.
stacked mode	A combination of image and overlay planes in which a single display has two "logical" screens, one the image planes, the other the overlay planes. Typically, the image planes are used for graphics while the overlay planes are used for text.
combined mode	A combination of image and overlay planes in which a single display has a single screen that is a combination of the image and overlay planes. Combined mode is available only on the TurboSRX 3-D Display Controller (HP part number 98730A).

Table 7-1. Graphics Boards, Display Controllers, and Available Screen Modes.

With this display hardware ...	You can use these modes ...			
	Image	Overlay	Stacked	Combined
HP 98542A	√			
HP 98543A	√			
HP 98544B	√			
HP 98545A	√			
HP 98547A	√			
HP 98548A	√			
HP 98549A	√			
HP 98550A	√	√	√	
HP 98720A	√	√	√	
HP 98730A	√	√	√	√

Syntax for 'X*screens' File Lines

The syntax for each line of an `X*screens` file is as follows:

```
/dev/device [ { [ depth { 8 } ]          } ]
            [ {   { 24 }  ]              }
            [ {   doublebuffer ]         }
            [ { depth 16 doublebuffer }  }
            [ monitorsize { numberin }   ]
            [             { numbermm }   ]
                                          [# comment]
```

/dev/*device* Specifies the name of the device file that the X server should read for this display.

depth Specifies the number of image and overlay planes available to the server (one pixel per plane).

doublebuffer Specifies **double buffer.** Double buffering divides video memory into halves and displays one half while drawing the other.

	Double buffering is used specifically with graphics programs that double buffer their screen output. This avoids "flashing" during screen redraw.
depth 16 doublebuffer	Specifies the division of the image planes into two 8-bit, doublebuffered halves.
monitorsize	Specifies the size of the monitor in inches or millimeters.

Determining the Number of Screen Devices

Each line of the text lists a separate screen device (except in combined mode). A screen device can be either a physical device, the CRT screen, or the image planes or overlay planes of a physical device.

For example, if you have a system that includes two physical display screens, you should create an `X*screens` file that contains two lines, one for each physical screen. If you have one physical display screen that is divided into image and overlay planes, you should create an `X*screens` file that also contains two lines. However, the lines will be different.

Mouse Tracking with Multiple Screen Devices

The mouse pointer tracks from left to right. For multiple-plane configuration files, the order of entry determines the tracking order of the mouse pointer. The first line in the file is the device on which the pointer appears when you start X.

Other lines correspond to the screens to which the pointer moves when the mouse is moved to the right or left and the pointer moves off the side of the current screen.

Thus the order of the lines is important because it tells the server to which screen to move the pointer.

Making a Device Driver File

When you specify a device in a screen configuration file, include the path, usually `/dev`. The device you specify must correspond to a device file in that path. If you don't have the appropriate device file, you must make that device using the HP-UX `mknod` command. See the "Creating Device Files" section of *HP-UX System Administration Manual* for information on `mknod`.

Examples

The following example shows how you might customize several **X*screens** files.

Suppose you have a high-resolution (1280×1024) TurboSRX screen connected to your S300. The image plane of this screen is accessed by the device file **/dev/crt**. The overlay plane is accessed by the device file **/dev/ocrt**. You would like to switch between three different screen configurations:

- One screen with X11 running in the image planes (image mode).
- Two screens with Starbase running in the image planes and X11 running in the overlay planes (stacked mode).
- One screen with Starbase running in the image plane and X11 running in the overlay plane (combined mode). The image planes have a depth of 24; the overlay planes have a depth of 8.

To accomplish this, you need to have the following three screen configuration files:

X0screens Containing the line:

```
/dev/crt
```

X1screens Containing the lines:

```
/dev/crt
/dev/ocrt
```

X2screens Containing the line:

```
/dev/crt /dev/ocrt depth 24 depth 8
```

Note that the first file is the default screen configuration provided with X11. The other two files must be created for the particular situation. The **X1screens** file contains two lines, one for each set of planes. Think of the lines as "stacked" for stacked mode. The **X2screens** file contains one line specifying both planes. Think of the lines as "combined" for combined mode.

An alternate means of achieving the same end is to modify the `XOscreens` file (or create a custom file) to contain the following lines:

```
### Default Configuration ###
/dev/crt

### Stacked Screens Mode ###
# /dev/crt
# /dev/ocrt

### Combined Screens Mode ###
# /dev/crt /dev/ocrt depth 24 depth 8
```

Using this method, the name of the file never changes from `XOscreens`. Instead, you edit the file, commenting out lines you don't want and uncommenting lines you do want, to switch screen devices.

Either method will work.

Defining Your Display

The DISPLAY environment variable establishes the host, display number, and screen number to which a system sends bitmapped output. For example, on Series 300 and Series 800 systems, the console is typically display 0, screen 0 by default and output is usually sent there.

However, most X11 clients have a `-display` option that enables you to specify a different host, display number, and screen number on which the client should display its output. (By default, clients display on the system on which they are started.)

Specifying a Display

As mentioned, you can specify a display for your local system or for individual clients.

The display for your local system, the DISPLAY environment variable, is part of your system environment and is used by all clients started and displayed locally.

Setting DISPLAY with 'x11start'. When you start X11, the `x11start`command sets the DISPLAY variable to the hostname of your system, display number 0, screen 0 with the following line:

```
: $ {DISPLAY:='hostname':0.0}; export DISPLAY
```

You can modify this line to set the DISPLAY variable to a different hostname or to the local "socket" facility by substituting 'local' for 'hostname' in the command line. You can also change either the display number or the screen number by substituting a different number in the above line.

The Difference Between 'local' and 'hostname'

Whether you choose `hostname` or `local` for the your DISPLAY variable is a matter of performance and depends on your situation. Using the hostname of your local system is the more traditional way of specifying the DISPLAY variable. However, using the local socket facility enables you to achieve faster communication between clients and server when both are on the same system.

Finding the DISPLAY Variable

You can view the current setting of your system's DISPLAY environment variable, by typing the following command line:

```
env Return
```

This causes a list similar to the following to display. The list contains the current environment settings for your system. Look for the DISPLAY setting.

```
DISPLAY=local:0.0
HOME=/users/alex
LOGNAME=alex
MAIL=/usr/mail/alex
PATH=/usr/bin/X11:.:./bin:../bin:/bin:/usr/local/bin:/usr/bin:/etc:
SHELL=/bin/csh
TERM=98720
TZ=PST8PDT
WINDOWID=7340052
```

In this example, the DISPLAY variable is set to "local:0.0."

Resetting the DISPLAY Variable

To reset the DISPLAY variable, do one of the following:

- If you use `csh`, type the following:

  `setenv DISPLAY` *host:display:screen* Return

- If you use `sh` or `ksh`, type the following:

  `DISPLAY=host:display.screen` Return

Remember whether you use a hostname or `local` for the DISPLAY variable is a matter of personal preference. Generally, if you do most of your processing locally, you'll find `local` more efficient.

For example, before you started X11 using your custom `X1screens` configuration file, you would issue the following command line for a `csh`:

```
setenv  DISPLAY  local:1.0  Return
```

Making 'X*.hosts' Files for Special Configurations

The default screen configuration file `X0screens` uses the default X11 remote host file `X0.hosts`. The `X0.hosts` file, you will recall, contains a list of all X11 hosts permitted to access your local server.

Each custom `X*screens` file requires that you make a special `X*.hosts` file. The number represented by the * is what causes the correct screen and host files to be used together.

Creating an 'X*.hosts' File

An `X*.hosts` file is an ASCII text file containing the hostnames of each remote host permitted to access your local server. The syntax is as follows:

```
host
host
host
```

For example, if you were on a network and regularly ran clients on hpcvfaa, hpcvfcc, and hpcvfdd, you would want the following lines in your `X*.hosts` file:

```
hpcvfaa
hpcvfcc
hpcvfdd
```

Note that aliases work as well as hostnames, provided that they are valid, that is, commonly known across the network.

Using Special Input Devices

The X Window System has an input device file that the X server reads to find out what input devices it should open and attach to the display. The default input device configuration file is `X0devices`. You can find it in the `/usr/lib/X11` directory.

The Default 'X0devices' File

The default file shipped with the X Window System contains lines of text, but does not specify any input configuration. Rather, it assumes the default input configuration of one keyboard and one pointer.

If this is your configuration, you may not want to change the contents of the file for three reasons:

- Clients can request and receive the services of an input device *regardless* of whether the device is specified in a device configuration file. Thus, you need not change the `X0devices` file, or create a custom file, even though you have a custom input configuration.
- Non-clients (terminal-based programs) such as Starbase *cannot* receive the services of an input device if the device is specified in the device configuration file. Any device in the device configuration file is opened for use by the X server. Thus, changing the `X0devices` file, or creating a custom file, in order to inform the server about a certain input device may interfere with a non-client's ability to access the device.

- Even if you have other *screen* configurations, you can rely on the default input device configuration without having to create an `X*devices` file to match every `X*screens` file. For example, if you had a custom `X8screens` file, you would not necessarily need an `X8devices` file.

A custom `X*devices` file is required only when you want to tell the *X server* about a custom input device configuration.

How the Server Chooses the Default Keyboard and Pointer

Input devices attach to HP computers through an interface known as the Hewlett-Packard Human Interface Link (HP-HIL). Up to seven input devices can be attached to each HP-HIL. However, if the `X*devices` file does not exist, or does not specify otherwise, the X server recognizes only two devices, a pointer and a keyboard (clients, however, may still recognize other input devices).

The X server uses the following order when choosing a pointer:

1. If the `X*devices` file specifies an input device as the pointer, the X server uses that device as the pointer.
2. If `X*devices` makes no specification, or there is no `X*devices` file, the X server takes the last mouse on the HP-HIL (the mouse farthest from the computer) as the pointer.
3. If the X server can open no mouse, it takes the last pointer device (knob box, graphics tablet, trackball, or touchscreen) on the HP-HIL as the pointer.
4. If the X server can open no pointer device, it takes the last keyboard on the HP-HIL as the pointer as well as the keyboard.
5. If no pointer can be opened, the server will not run.

The X server uses a similar order when determining the keyboard:

1. If the `X*devices` file specifies an input device as the keyboard, the X server uses that device as the keyboard.
2. If `X*devices` makes no specification, or there is no `X*devices` file, the X server takes the last keyboard on the HP-HIL (the keyboard farthest from the computer) as the keyboard.

3. If the X server can open no keyboard, it takes the last key device (buttonbox, barcode reader) on the HP-HIL as the keyboard.
4. If no keyboard can be opened, the server will not run.

Creating a Custom 'X*devices' File

At some point, you may want to instruct the server to open a particular device as the keyboard or pointer or have the server open another input device as an extension of the keyboard or pointer. Additional devices with keys are treated as extensions to the keyboard; additional devices that point are treated as extensions to the pointer.

To tell the server about a non-default input device configuration, you must add a device specification line to the appropriate `X*devices`. For example, you would use `X0devices` if you used `X0screens` and `X2devices` if you used `X2screens`.

`X*devices` files are ASCII text files. You can use any ASCII text editor to modify them. Similar to `X*screens` files, you add a device line to the file for each input device you want the server to know about.

Syntax

The device specification lines that you add to the `X*devices` file can have either of two syntaxes.

The Syntax for Device Type and Relative Position. The following syntax uses device type and relative position on the HP-HIL to specify input devices:

relativeposition devicetype use [# *comments*]

relativeposition	Specifies the position of a device on the HP-HIL relative to other input devices of the same kind.
devicetype	Specifies the type of input device.
use	Specifies whether the device is the keyboard, the pointer, or has some other use.

Separate the parts of your entry with tabs or spaces. The position of an input device on the HP-HIL is relative to other devices of that type. Thus, `first` means the device connected closest to the computer on the HP-HIL of any device *of that type.*

This syntax is most useful for systems running a single X server with no other programs directly opening input devices. Here, if you add a new input device to the HP-HIL, you don't have to edit the `X*devices` file unless the device is of the same type as one already named in the file and you add the new device ahead of that existing device.

This syntax may become ambiguous, however, when more than one X server is running on the same system or when non-client programs directly access input devices. This is because `first` actually means first device of that type *available* to the server. Thus, a device may be physically first on the HP-HIL, but not first for the server if the device is unavailable because it is currently being used by some other program.

The Syntax for Device File Name. The following syntax uses the device file name to specify input devices:

/path/devicefile *use* [# *comments*]

/path/devicefile	Specifies the path and device file to use as the input device.
use	Specifies whether the device is the keyboard, the pointer, or has some other use.

This syntax is unambiguous when several X servers are running on the same computer or when non-client programs directly access the input device.

The Syntax for Reconfiguring the Path to Device Files. The default path to the device files is `/dev`, but you can specify another path if you choose. Also, if you have more than one HP-HIL, you can specify which HP-HIL the server should use. This is useful on S800 systems with more than one HP-HIL.

The syntax for this is as follows:

path `hil_path` [# *comments*]

path Specifies the path to the device files.

The path specified is handled differently depending on whether your system is an S300 or an S800. S300s only have a single HP-HIL, so no matter what path you specify, the server always checks *only that path* HP-HIL for input devices to open. For example, if you specify **/tmp/foo hil_path** in your **X*devices** file, the server would attempt to open devices **/tmp/foo1** through **/tmp/foo7** on the HP-HIL.

However, S800s can have up to four HP-HILs numbered 0 through 3. For the S800s, the HIL number you specify is the *only* place the server checks for input devices to open. Thus if you specify **/dev/hil_2 hil_path**, the server would try to open input devices 1 through 7 on HP-HIL 2 and, if that failed, would not open any input devices at all. If you specified a path that did not end with a digit in the range 0-3, the server would search four HP-HILs for devices to open using that path as the root of the device file names. For example, if you specified **/tmp/foo hil_path**, the server would attempt to open decive files **/tmp/foo0.1** through **/tmp/foo0.7**, **/tmp/foo1.1** through **/tmp/foo1.7**, and so on through **/tmp/foo3.7**

Selecting Values for 'X*devices' Files

`X*devices` files use the following special names for positions, devices, and uses:

Table 7-2. Special Names for 'X*devices' Files.

Positions	Device Name	Device Type	HP Part Number	Uses
first	keyboard	keyboard	46021A*	keyboard
second	mouse	pointer	46060A*	pointer
third	tablet	pointer	46087A*	other
fourth	buttonbox	keyboard	46086A*	
fifth	barcode	keyboard	92916A*	
sixth	one_ knob	pointer	46083A*	
seventh	nine_ knob	pointer	46085A*	
	quadrature	pointer	46094A*	
	touchscreen	pointer	35723A*	
	trackball	pointer	80409A*	
	null			
* or equivalent				

Note that the nine-knob box appears to the X server as three separate input devices. Each row of knobs is a separate device with the first device being the bottom row.

Note also that the HP barcode reader has two modes: keyboard and ASCII. The modes are set via switches on the reader. If you set the barcode reader to ASCII mode, it appears to the server as a barcode reader and the device name is therefore `barcode`. However, if you set the barcode reader to emulate a keyboard, the barcode reader *appears as a keyboard* and the device name should therefore be `keyboard`. What distinguishes a barcode reader set to keyboard mode from a real keyboard is the relative position or the device filename, depending on which syntax you use.

Similar to the barcode reader, the trackball appears to the server, not as a trackball, but *as a mouse.* Therefore, to specify a trackball, use the `mouse` device name. Again, what specifies the trackball instead of the real mouse is the relative position or the device filename, depending on which syntax you use.

Configuring an Output-Only X Window System

You can create a system on which the X server runs, but which does not have any input devices. In this case, clients could be run from a remote terminal, or from a remote host, and their output directed to the X server.

To create an X11 system with no input, include the following lines in the `X0devices` file:

```
first null   keyboard
first null   pointer
```

Examples

Suppose your input devices include a graphics tablet, a keyboard, and another graphics tablet and you want to use the tablet closest to the computer as the pointer. You would have the following lines in your `X*devices` file:

```
first    tablet     pointer     The pointer.
second   tablet     other       An extension of the pointer.
first    keyboard   keyboard    The keyboard.
```

In this example, input from the second graphics tablet would appear as an extension of the input from the pointer device, the first tablet.

Now suppose you add another keyboard and a barcode reader (set to ASCII mode) to the above configuration, and you want the keyboard farthest from the computer to be the keyboard device with the barcode reader serving as an extension to it. You would have the following lines in your `X*devices` file:

```
first    tablet     pointer     The pointer.
second   tablet     other       An extension of the pointer.
first    keyboard   other       An extension of the keyboard.
second   keyboard   keyboard    The keyboard.
first    barcode    other       An extension of the keyboard.
```

The example now includes input from the first keyboard and from the barcode reader as extensions of the input from the second keyboard. To the X server, the input is indistinguishable.

Note that the barcode reader is in ASCII mode in this example. If the barcode reader were in keyboard mode, the last line of the example should read as follows:

```
third keyboard    other
```

In keyboard mode, the barcode reader is merely the third keyboard on the HP-HIL.

Now suppose you add a nine-knob box to the configuration, but only use the first two rows of knobs. You would have the following lines in the input device configuration file (assuming the barcode reader is set to ASCII mode):

```
first     tablet      pointer     The pointer.
second    tablet      other       An extension of the pointer.
first     keyboard    other       An extension of the keyboard.
second    keyboard    keyboard    The keyboard.
first     barcode     other       An extension of the keyboard.
first     nine_knob   other       Bottom row, pointer extension.
second    nine_knob   other       Middle row, pointer extension.
```

Note that specifying an `other` input device in an `X*devices` file has certain consequences. Each input device you specify as `other` in `X*devices` is opened *exclusively* by the X server. This means that the device is available for clients, but *is not available* for direct access by non-client programs. Since it isn't necessary to list `other` devices in `X*devices` for clients to access them, it may be better for you to omit `other` devices from your `X*devices` file. Include them only if no Starbase or other non-client programs access them directly.

Going Mouseless with the 'X*pointerkeys' File

Your work situation may lack sufficient desk space to adequately use a mouse pointer. You may, therefore, want to "go mouseless" by naming the keyboard (or some other input device) as the pointer.

To go mouseless, you need to have the proper configuration specified in the `X*devices` file and to have a special configuration file named `X*pointerkeys`. The default `X*pointerkeys` file is named `X0pointerkeys`. You can find it in the `/usr/lib/X11` directory. In light of your experience with `X0sceens` and `X0devices`, you will probably recognize this as no mere coincidence.

The `X*pointerkeys` file enables you to specify the following:

- The keys that move the pointer.
- The keys that act as mouse buttons.
- The increments for movement of the pointer.
- The key sequence that resets X11.
- The pixel threshold that must be exceeded before the server switches pixel planes in stacked screen mode.

Configuring 'X*devices' for Mouseless Operation

If you have only a keyboard and no mouse on the HP-HIL, and you want the keyboard to serve as both keyboard and pointer, you don't have to change the default configuration of `X0devices`. The default input device configuration automatically assigns the pointer to the keyboard if a pointer can't be opened by the server.

If you have two input devices, you may need to explicitly specify which device should be the keyboard and which the pointer.

The Default Values for the 'X*pointerkeys' File

By default, when you configure your keyboard as the pointer, the X server chooses certain of the number pad keys and assigns them mouse operations. Some number pad keys are assigned to pointer movement; other number pad keys are assigned to button operations.

If you don't need to change the pointer keys from their default specifications, you don't need to do anything else to use your keyboard as both keyboard and pointer. However, if you need to change the default pointer keys, you must edit the `X0pointerkeys` file or create a new `X*pointerkeys` file. The `X*pointerkeys` file is the file that specifies which keys are used to move the pointer when you use the keyboard as the pointer.

The default key assignments are listed in the tables in the following section on customizing the `X*pointerkeys` file.

Creating a Custom 'X*pointerkeys' File

You need to modify the existing `X0pointerkeys` file only if the following statements are true:

- You want to use the keyboard for a mouse.
- You want to change the pointerkeys from their default configuration.
- You use the `X0screens` file to configure your display.

You need to create a custom `X*pointerkeys` file only if the following statements are true:

- You want to use the keyboard for a mouse.
- You want to change the pointerkeys from their default configuration.
- You use a configuration file other than the `X0screens` file to configure your display.

Syntax

You assign a keyboard key a mouse function (pointer movement or button operation) by inserting a line in the `X*pointerkeys` file. One line for each action. Lines in the `X*pointerkeys` file have the following syntax:

function keyname [# comment]

Assigning Mouse Functions to Keyboard Keys

You can assign any mouse function, either a pointer movement or a button operation, to any keyboard key. However, you should first make sure that the key you are thus changing the meaning of doesn't already serve some absolutely vital function.

You can assign keyboard keys to pointer directions by specifying options in a **X*pointerkeys** file. The following table lists the pointer movement options, the **X*pointerkeys** functions that control them, and their default values:

Table 7-3. Pointer Movement Functions.

To do this ...	Use this function ...	The default key is ...
Move the pointer to the left.	**pointer_left_key**	keypad_1
Move the pointer to the right.	**pointer_right_key**	keypad_3
Move the pointer up.	**pointer_up_key**	keypad_5
Move the pointer down.	**pointer_down_key**	keypad_2
Add a modifier key to the pointer direction keys.	**pointer_key_mod1**	no default
Add a second modifier key to the pointer direction keys.	**pointer_key_mod2**	no default
Add a third modifier key to the pointer direction keys.	**pointer_key_mod3**	no default

Note that the pointer direction keys are the key*pad* number keys on the right side of the keyboard, not the key*board* number keys above the text character keys.

You can assign keyboard keys to pointer distances by specifying options in a **XOpointerkeys** file. The following table lists the options that determine the distance of pointer movements, the **X*pointerkeys** functions that control them, and their default value:

Table 7-4. Pointer Distance Functions.

To do this ...	Use this function ...	The default value is ...
Move the pointer a number of pixels.	**pointer_move**	10 pixels
Move the pointer using a modifier key.	**pointer_mod1_amt**	40 pixels
Move the pointer using a modifier key.	**pointer_mod2_amt**	1 pixel
Move the pointer using a modifier key.	**pointer_mod3_amt**	5 pixels
Add a modifier to the distance keys.	**pointer_amt_mod1**	no default
Add a modifier to the distance keys.	**pointer_amt_mod2**	no default
Add a modifier to the distance keys.	**pointer_amt_mod3**	no default

You can assign keyboard keys to mouse button operations by specifying options in a `X*pointerkeys` file. The following table lists the button operations, the `X*pointerkeys` functions that control them, and their default values:

Table 7-5. Button Operation Functions.

To do this ...	Use this function ...	The default key is ...
Perform button 1 operations.	`pointer_button1_key`	keypad_ *
Perform button 2 operations.	`pointer_button2_key`	keypad_ /
Perform button 3 operations.	`pointer_button3_key`	keypad_ +
Perform button 4 operations.	`pointer_button4_key`	keypad_ -
Perform button 5 operations.	`pointer_button5_key`	keypad_ 7

You can change the key sequence that exits the X Window System. Also, if you use both image and overlay planes, you can change the distance you must move the pointer before you switch planes. The following table lists these options, the `X*pointerkeys` functions that control them, and their default values:

Table 7-6. Reset and Threshold Functions.

To do this ...	Use this function ...	The default key is ...
Exit the X Window System	`reset`	break
Add a modifier to the exit key.	`reset_mod1`	control
Add a modifier to the exit key.	`reset_mod2`	left_ shift
Add a modifier to the exit key.	`reset_mod3`	no default
Set the threshold for changing between image and overlay planes in stacked mode.	`screen_change_amt`	30 pixels

The `screen_change_amt` enables you to avoid switching from one set of pixel planes to another if you accidentally run the pointer off the edge of the screen. The `screen_change_amt` option establishes a "distance threshold" that the pointer must exceed before the server switches pixel planes. As the above table shows, the default width of the threshold is 30 pixels, but acceptable values range from 0 to 255.

Examples

For example, a common change that you can easily make is to change the keyboard's [▲], [▼], [◀], and [▶] keys to be the pointer direction keys. Press the [▲] key and the pointer moves up. This makes perfect sense. Or at least it does until you try to move the text cursor in your `hpterm` window. If you reassign the arrow keys to the pointer, they will no longer work for the cursor.

Fortunately, the X Window System enables you to pick up to three keys from among the two [Shift] keys, the two [Extend char] keys, and the [CTRL] key and use them each as a **modifier key.** A modifier key is a key that, when you hold it down and press another key, changes the meaning of that other key.

Modifier keys in the `X*pointerkeys` file have three functions:

- They specify that a certain operation can't take place until they are pressed.
- They enable you to adjust the distance covered by the pointer during a movement operation.
- They enable you to change the key sequence that exits you from X11.

For example, you can overcome the problem in the last example by assigning the [Left Shift] key as a modifier to the pointer direction keys. Now, to move the *cursor* to the right, you press [▶] as usual. To move the *pointer* to the right, you press [Left Shift] [▶].

Specifying Pointer Keys

The following table lists the valid keynames to use when assigning keyboard keys to mouse functions:

Table 7-7. Valid Pointer Keynames.

Typewriter Keys:					
1	A	K	U	left_ shift	return
2	B	L	V	left_ extend	,
3	C	M	W	-	.
4	D	N	X	=	/
5	E	O	Y	backspace	right_ shift
6	F	P	Z	[	space_ bar
7	G	Q	‘	]	right_ extend
8	H	R	tab	\	
9	I	S	caps_ lock	;	
0	J	T	control	’	
Function Keys:					
f1	f3	f5	f7	blank_ f9	blank_ f11
f2	f4	f6	f8	blank_ f10	blank_ f12
Keypad Keys:					
keypad_ 1	keypad_ 4	keypad_ 7	keypad_ 0	keypad_ +	keypad_ comma
keypad_ 2	keypad_ 5	keypad_ 8	keypad_ *	keypad_ -	keypad_ tab
keypad_ 3	keypad_ 6	keypad_ 9	keypad_ /	keypad_ enter	keypad_ period
Special Keys:					
enter	stop	clear_ line	delete_ line	home_ cursor	cursor_ up
escape	menu	clear_ display	insert_ char	prev	next
break	system	insert_ line	delete_ char	select	cursor_ left
cursor_ down	cursor_ right				

Examples

If you only have one keyboard and no mouse, and you can live with the default pointer key assignations, you don't have to do anything else to configure your system for mouseless operation. To move the pointer to the left 10 pixels, you would press the 1 key on the keypad. To press mouse button 1, the select button, you would press the * key on the keypad.

However, suppose you wanted to move only one pixel to the left. Although the default value of `pointer_mod2_amt` is one pixel, no key is assigned to the modifier for that amount. Thus, you would need to edit the `X0pointerkeys` file (or create an `X*pointerkeys`) to include a line assigning one of the modifier keys to `pointer_amt_mod2`. The following line in `X0pointerkeys` assigns the Left Shift key to `pointer_amt_mod2`:

```
###pointerfunction      key
pointer_amt_mod2     left_shift
```

Or suppose you wanted to set up your `X0pointerkeys` file so that you could move 1, 10, 25, and 100 pixels. The following lines show one way to specify this:

```
###pointer function      key
pointer_amt_mod1      left_extend
pointer_amt_mod2      left_shift
pointer_amt_mod3      control
pointer_move          1_pixels
pointer_mod1_amt      10_pixels
pointer_mod2_amt      25_pixels
pointer_mod3_amt      100_pixels
```

With these lines in effect, one press of the 1 key on the keypad moves the pointer 1 pixel to the left. Pressing the left Extend char and 1 moves the pointer 10 pixels to the left. Pressing Left Shift 1 moves the pointer 25 pixels to the left. And pressing CTRL 1 moves the pointer 100 pixels to the left.

Or, take the case previously mentioned where you want to use the arrow keys for both text cursor and mouse pointer. You could insert the following lines in your `X0pointerkeys` file:

```
###pointer function      key
pointer_key_mod1       left_shift
pointer_left_key       cursor_left
pointer_right_key      cursor_right
pointer_up_key         cursor_up
pointer_down_key       cursor_down
```

The above lines enable you to use the arrow keys for cursor movement, while using the shifted arrow keys for pointer movement. Note that it is the left Shift key only (not the right Shift) that modifies the press of an arrow key from cursor to pointer movement.

Now take this scenario a step further. Suppose you want to use the arrow keys to operate the pointer, and you also need the arrow keys to control the cursor in an `hpterm` window, but, as luck would have it, another program you frequently operate uses the shift-arrow key sequence to control its cursor.

The easiest way to solve this dilemma is to call in another modifier. The following lines illustrate this. Compare them to the previous example.

```
###pointer function      key
pointer_key_mod1       left_shift
pointer_key_mod2       left_extend
pointer_left_key       cursor_left
pointer_right_key      cursor_right
pointer_up_key         cursor_up
pointer_down_key       cursor_down
```

In this example,

- Pressing the ▲ key moves the `hpterm` *text cursor* up.
- Pressing Shift ▲ moves the *cursor* up in the program you frequently operate.
- Pressing Shift Extend char ▲ moves the *pointer* up.

Using a similar technique, you can also reassign the CTRL Left Shift Reset sequence that exits X11. You can specify the press of a single key to exit X11,

or a combination of two, three, or four key presses. Just make sure that the key sequence you select isn't something you're going to type by accident.

Customizing Keyboard Input

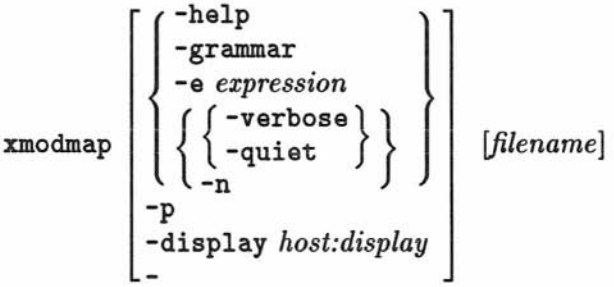
Besides remapping the mouse's pointer and buttons to your keyboard, you can remap any key on the keyboard to any other key.

Modifying Modifier Key Bindings with 'xmodmap'

To change the meaning of a particular key for a particular X11 session, or to initialize the X server with a completely different set of key mappings, use the **xmodmap** client.

Syntax and Options

The syntax for **xmodmap** is as follows:



-display	Specifies the host, display number, and screen to use.
-help	Displays a brief description of **xmodmap** options.
-grammar	Displays a brief description of the syntax for modification expressions.
-verbose	Prints log information as **xmodmap** executes.
-quiet	Turns off verbose logging. This is the default.

-n Lists changes to key mappings without actually making those changes.

-e Specifies a remapping expression to be executed.

-p Prints a list of current key mappings to the standard output.

- Specifies that the standard input should be used for input file.

filename Specifies a particular key mapping file to be used.

Specifying Key Remapping Expressions

Whether you remap a single key "on the fly" with a command-line entry or install an entire new keyboard map file, you must use valid expressions in your specification, one expression for each remapping.

A valid expression is any one of the following:

Table 7-8. Valid 'xmodmap' Expressions.

To do this ...	Use this expression ...
Assign a key symbol to a keycode.	`keycode` *keycode* = *keysym*
Replace a key symbol expression with another.	`keysym` *keysym* = *keysym*
Clear all keys associated with a modifier key.	`clear` *modifier*
Add a key symbol to a modifier.	`add` *modifier* = *keysym*
Remove a key symbol from a modifier.	`remove` *modifier* = *keysym*

keycode Refers to the numerical value which uniquely identifies each key on a keyboard. Values may be in decimal, octal, or hexadecimal.

keysym Refers to the character symbol associated with a keycode.

modifier Specifies one of the eight modifier names.

The keycodes and key symbols differ from keyboard to keyboard. For a list of the valid keycodes for your keyboard, use the **xprkbd** command explained below.

The following are the modifier names available for use in keyboard customization:

Table 7-9. Valid Modifier Names.

Modifier Names			
Shift	Control	Mod2	Mod4
Lock	Mod1	Mod3	Mod5

On Hewlett-Packard keyboards, the `mod1` modifier is set to the Extend char keys (Meta_ L and Meta_ R). However, any of the modifiers can be associated with any valid key symbol. Additionally, although you can't associate more than one key symbol with a modifier, you can associate more than one modifier with a key symbol.

For example, you can press d to print a lower case "d", Shift d to print a capital "D", Extend char d to print something else, and Shift Extend char d to print still something else.

The `xmodmap` client gives you the power to change the meaning of any key at any time or to install a whole new key map for your keyboard. Like remapping the mouse, a little forethought goes a long way.

Examples

Suppose you had the unfortunate habit of hitting the Caps key at the most inopportune moments. You could remove the Caps lock key from the lock modifier, swap it for the f1 key, then map the f1 key to the lock modifier. The way to do this is by creating a little swapper file that contains the following lines:

```
!This file swaps the [Caps] key with the [F1] key.

remove Lock = Caps_Lock
keysym Caps_Lock = F1
keysym F1 = Caps_Lock
add Lock = Caps_Lock
```

Note the use of the ! in the file to start a comment line. To put your "swapper" file into effect, enter the following on the command line:

```
xmodmap swapper Return
```

If you use such a swapper file, you should probably have an unswapper file. The following file enables you to swap back to the original keyboard mapping without having to exit X11:

```
!This file unswaps the [F1] key with the [Caps key.

remove Lock = Caps_Lock
keycode 0x54 = F1
keycode 0x37 = Caps_Lock
add Lock = Caps_Lock
```

Note the use of the hexadecimal values to reinitialize the keycodes to the proper key symbols. You put your "unswapper" file into effect by entering the following command line:

```
xmodmap unswapper Return
```

On a larger scale, you can change your current keyboard to a Dvorak keyboard by creating a file with the appropriate keyboard mappings. Typically, you would keep this as a special file in your home directory, giving it some name like ".keymap." The easiest way to install your Dvorak keyboard map is by including a line in your `.x11start` file like the following:

```
xmodmap .keymap
```

Printing a Key Map with 'xprkbd'

The `xprkbd` client prints a list of the key mappings for the current keyboard.

The file contains up to four 3-part columns. The first column contains unmodified key values, the second column contains shifted key values, the third column contains meta (Extend char) key values, and the fourth column contains shifted meta key values.

Additionally, each column is in three parts: hexadecimal keycode value, hexadecimal key symbol value, and key symbol name.

Syntax and Options

The syntax for **xprkbd** is as follows:

xprkbd [**-display** *host:display*] [**-help**]

-display Specifies the host, display number, and screen to use.

-help Displays a brief description of **xprkbd** options.

Creating a Custom Color Database with 'rgb'

Depending on your needs, you may want to make your own custom color database modeled after the **rgb.txt** file. The **rgb.txt** file is the source file the **rgb** client uses to compile the two files the server uses for color database information. Thus, if you listed the files in **/usr/lib/X11** that began with **rgb***, you'd find not only **rgb.txt**, but the two files **rgb.dir** and **rgb.pag**.

The file **rgb.txt** is the default color data base for the X Window System. The file is an ASCII text file and, in case you haven't looked at it, it contains four columns: red value, green value, blue value, and color name. The following lines are from **rgb.txt**. Note that the red, green, and blue values are given as the decimal equivalents of their hexadecimal values.

Table 7-10. Some Lines from 'rgb.txt'.

Red	Green	Blue	Color Name
47	47	100	MidnightBlue
35	35	117	navy blue
35	35	117	NavyBlue
35	35	117	navy
35	35	117	Navy
114	159	255	sky blue
114	159	255	SkyBlue

As the above lines illustrate, several lines are sometimes necessary to account for alternate spellings of the same color.

To make a custom color database, start your ASCII text editor and open a new file for your database. Another option is to make a copy of `rgb.txt`, giving it a new name, and add, edit, or delete those values.

Use the following steps to add entries to the database:

1. Specify a decimal value for the red aspect of the color.
2. Press the spacebar or Tab.
3. Specify a decimal value for the green aspect of the color.
4. Press the spacebar or Tab.
5. Specify a decimal value for the blue aspect of the color.
6. Press the spacebar or Tab.
7. Specify a color name for the color.
8. Press Return.
9. Repeat steps 2-8 for the other colors in your custom color database.
10. Save your new database file and exit the editor.

To get the other two files, the ones used by the server, use the **rgb** client. The **rgb** client has the following syntax:

rgb *outfile* <*infile*

where *infile* is the name of your custom database, the text file you created. The **rgb** client will create *outfile*.**dir** and *outfile*.**pag**. Note that, if you choose to modify the existing **rgb.txt** (at least make a backup copy if you do), you must run it through the **rgb** client before any changes take effect.

To put your new color database into effect, you must add it to your **x11start** command line. For example, if your new database is composed of the files **2brite.txt**, **2brite.dir**, and **2brite.pag**, type the following command line to start your X environment:

```
x11start -- -co 2brite Return
```

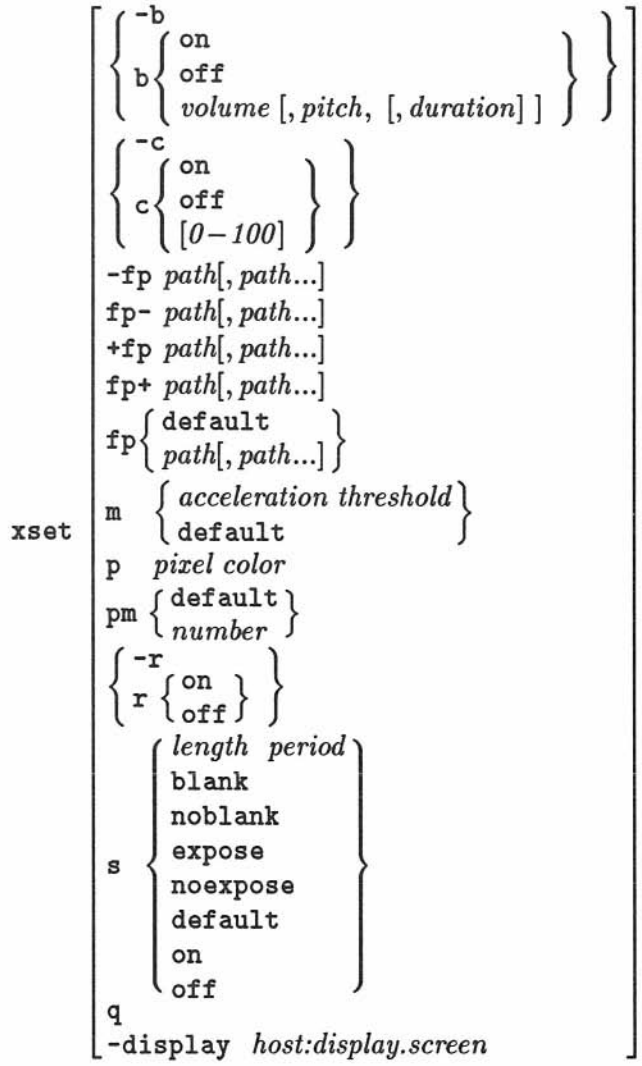
The server by default looks in the **/usr/lib/X11** directory for information and this example assumes that that is where your **2brite*** files are.

Changing Your Preferences with 'xset'

The **xset** client allows you to change your preferences for display options. These preferences last for the length of the X Window System session.

Syntax and Options

The syntax for **xset** is as follows:



b on/off — Turns the bell on or off.

b v[p[d]] — Specifies the bell volume, pitch, and duration. *Volume* is a percentage between 0 and 100 and can be specified without specifying pitch and duration. *Pitch* is a number of hertz and is specified together with a volume. *Duration* is a number of

Compiling Bitmap Distribution Fonts into Server Natural Format

The X Window System fonts that you select and that the server then uses to display text can be in either of two font formats: server compressed format (scf) or server natural format (snf). Both formats function the same. However, the compressed format takes up less storage space on disk but must be uncompressed by the server for use.

Which format a font is in is signified by the extension that appears after the font name listed in **/usr/lib/X11/fonts**. A **.scf** signifies a compressed format; a **.snf** signifies a natural (uncompressed) format.

You can compress a snf font file using the HP-UX **compress** command. You can uncompress a scf font file using the HP-UX **uncompress** command.

The X Window System includes a font compiler, **xfc**, which enables you to convert a font in bitmap distribution format (BDF 2.1) into server natural format.

Syntax and Options

The syntax for **xfc** is as follows:

xfc [**-p** *number*] [{ **-l** }] *filename* **>***font*.**snf**
[{ **-m** }]

-p — Specifies that font characters should be padded on the right with zeros to the boundary of word *number* where *number* is 1, 2, 4, or 8.

-l — Specifies the output of **xfc** to be least significant byte first.

-m — Specifies the output of **xfc** to be most significant byte first.

filename — Specifies the name of the bitmap distribution format font to convert to server natural format.

Note that **xfc** by default sends output to standard output (typically the screen). To capture the output as a **.snf** file, therefore, you must redirect the output as shown in the above syntax.

Example

The following example takes the font file **timrom12b.bdf**, a bitmap distribution file, converts it to an snf file, then compresses it into an scf file:

```
xfc  timrom12b.bdf  > timroman12b.snf  Return
compress  timrom12b.snf  Return
mv  timrom12b.snf.Z  timrom12b.scf  Return
```

Note that the **compress** command does not willingly create a **.scf** file, so you have to **mv** or **cp** the compressed **timrom12b.snf.Z** file to **timrom12b.scf**.

Using 'xrdb' to Configure the X Server

If you have no **.Xdefaults** file in your home directory, the **x11start** command uses the **xrdb** client to load **/usr/lib/X11sys.Xdefaults** into the server's RESOURCE_MANAGER property. The RESOURCE_MANAGER property contains the list of resources available to the server including the specifications for client colors, keyboard focus policy, and other, similar configurable resources.

You can use **xrdb** to load a custom resource configuration file into the server's RESOURCE_MANAGER property. One example of the potential that this has is that it enables you to load a resource file that contains conditional statements. You can define one set of resources for use in one particular situation while another set of resources is defined for a different situation.

The benefit of reading the file into the server, instead of having it on disk, is that the server doesn't have to keep reading the file each time you start a new client. Additionally, if you start remote clients and display them on your local screen, the clients will use your local colors. Without a **.Xdefaults** file on the remote host, the clients would appear on your display in their remote colors.

Compiling Bitmap Distribution Fonts into Server Natural Format

The X Window System fonts that you select and that the server then uses to display text can be in either of two font formats: server compressed format (scf) or server natural format (snf). Both formats function the same. However, the compressed format takes up less storage space on disk but must be uncompressed by the server for use.

Which format a font is in is signified by the extension that appears after the font name listed in **/usr/lib/X11/fonts**. A **.scf** signifies a compressed format; a **.snf** signifies a natural (uncompressed) format.

You can compress a snf font file using the HP-UX **compress** command. You can uncompress a scf font file using the HP-UX **uncompress** command.

The X Window System includes a font compiler, **xfc**, which enables you to convert a font in bitmap distribution format (BDF 2.1) into server natural format.

Syntax and Options

The syntax for **xfc** is as follows:

xfc [**-p** *number*] [{**-l**} {**-m**}] *filename* **>***font*.**snf**

-p — Specifies that font characters should be padded on the right with zeros to the boundary of word *number* where *number* is 1, 2, 4, or 8.

-l — Specifies the output of **xfc** to be least significant byte first.

-m — Specifies the output of **xfc** to be most significant byte first.

filename — Specifies the name of the bitmap distribution format font to convert to server natural format.

Note that **xfc** by default sends output to standard output (typically the screen). To capture the output as a **.snf** file, therefore, you must redirect the output as shown in the above syntax.

Example

The following example takes the font file **timrom12b.bdf**, a bitmap distribution file, converts it to an snf file, then compresses it into an scf file:

```
xfc  timrom12b.bdf  > timroman12b.snf  Return
compress  timrom12b.snf  Return
mv  timrom12b.snf.Z  timrom12b.scf  Return
```

Note that the **compress** command does not willingly create a **.scf** file, so you have to **mv** or **cp** the compressed **timrom12b.snf.Z** file to **timrom12b.scf**.

Using 'xrdb' to Configure the X Server

If you have no **.Xdefaults** file in your home directory, the **x11start** command uses the **xrdb** client to load **/usr/lib/X11sys.Xdefaults** into the server's RESOURCE_MANAGER property. The RESOURCE_MANAGER property contains the list of resources available to the server including the specifications for client colors, keyboard focus policy, and other, similar configurable resources.

You can use **xrdb** to load a custom resource configuration file into the server's RESOURCE_MANAGER property. One example of the potential that this has is that it enables you to load a resource file that contains conditional statements. You can define one set of resources for use in one particular situation while another set of resources is defined for a different situation.

The benefit of reading the file into the server, instead of having it on disk, is that the server doesn't have to keep reading the file each time you start a new client. Additionally, if you start remote clients and display them on your local screen, the clients will use your local colors. Without a **.Xdefaults** file on the remote host, the clients would appear on your display in their remote colors.

How Applications Get their Attributes

The X Window System uses multiple configuration files. An application can get its color and other attributes from several different files. Therefore, how an application gets its attributes (for example, its foreground color) might, on occasion, seem a little mysterious.

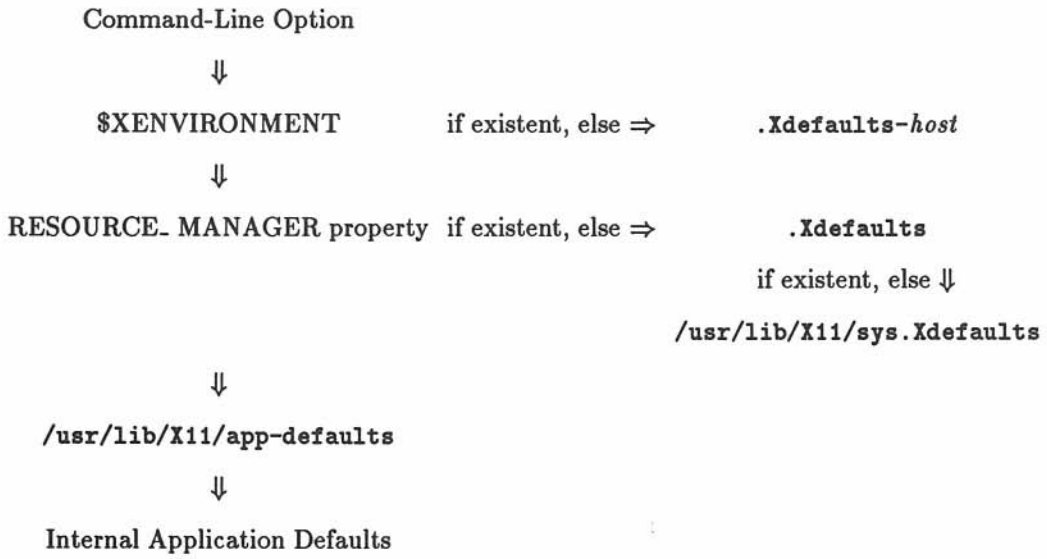
Where to Find Attributes

You can find application attributes specified in any of the following places:

- A command line may contain attribute options. These options are good for only that one instance of the application. A command-line option is the equivalent of a *client.resource* statement in a **.Xdefaults** file.
- The **XENVIRONMENT** environment variable, if present, may contain the name of a file that specifies application attributes.
- A **.Xdefaults-***host* file in your home directory may contain application attributes to be used for a specific remote host. A **.Xdefaults-***host* file is only read if no **XENVIRONMENT** variable exists.
- The RESOURCE_MANAGER property of the server may contain attributes from a file loaded into the server with **xrdb**.
- A **.Xdefaults** file in your home directory, or a **sys.Xdefaults** file in **/usr/lib/X11** may contain application attributes to be used on your local system. The **.Xdefaults** file is read only if no file has been loaded into the RESOURCE_MANAGER property. The **sys.Xdefaults** file is only read if no **.Xdefaults** file exists in your home directory.
- The **/usr/lib/X11/app-defaults** directory may contain application-specific configuration files that specify attributes for a particular application. An **app-defaults** file is the equivalent of a *Class*resource* statement in an **.Xdefaults** file.
- An application may have internal defaults that specify attributes when no resource configuration files exist.

The following figure presents the hierarchy of resource configuration files:

Figure 7-1. The Heirarchy of Resource Configuration Files.



Note that the figure is additive from top to bottom, but exclusive from left to right. In other words, if a resource isn't specified on the command line, it is added from the $XENVIRONMENT variable, if set, and if set, no resource specifications come from `.Xdefaults-`*host*. Likewise, the resource specification could come from the RESOURCE_ MANAGER property, and if so, no resource specifications come from `.Xdefaults`.

Class Struggle and Individual Identity

Additionally, the visible outcome of any attribute specification is influenced by whether that specification is for an individual resource or for a class of resources.

An individual resource begins with a lowercase letter. For example, `foreground` refers to the foreground resource. A class resource, however, begins with an uppercase letter. For example, `Foreground` refers to the the entire class of foreground resources.

Thus, if no other specifications overruled, the line `*foreground: blue` in your `.Xdefaults` file would make all foregrounds blue. However, the line `*Foreground: blue` would make all resources that belonged to the Foreground

class blue. This would include such resources as foreground; cursorColor; pointerColor; bottomShadowColor for softkeys, hpwm, icons, and mattes; clock hands; and highlight.

The Order of Precedence Among Attributes

In general, follow this rule of thumb to determine the effect of a resource specification:

A more detailed specification takes precedence over a less detailed specification.

For example, suppose you included the following lines in your `.Xdefaults` file:

```
*Foreground:            red
HPterm*Foreground:      DarkSlateGray
HPterm*foreground:      coral
HPterm*cursorColor:     green
```

The first line makes all resources of the class Foreground red. The second line overrules the first line, but *only* in the case of clients of class HPterm (of which there is only one - the `hpterm` client itself). Line two makes the Foreground class resources of all `hpterm` clients DarkSlateGray. Lines three and four give `hpterm` clients coral foregrounds and green cursors respectively, while the other resources of class Foreground (pointerColor, cursorColor, softkey foreground and bottomShadowColor, and scrollbar foreground and bottomShadowColor) remain DarkSlateGray.

Naming a Client

Additionally, you can give a client of some class a name. This allows you to allocate resources to that client by class, by client, *and* by name.

For example, `HPterm` is a client class. The `hpterm` window is a client of that class. But, using the following syntax, you can also give an `hpterm` client of the `HPterm` class a *name:*

client`.name:` *name*

Thus, you could add three lines to the lines above and have the following specfications in your `.Xdefaults` file:

```
*Foreground:            red
HPterm*Foreground:      DarkSlateGray
HPterm*foreground:      coral
HPterm*cursorColor:     green
hpterm.name:            BWterm    #Monochrome hpterm window
BWterm*Foreground:      black
BWterm*Background:      white
```

This illustrates the ability to create a named window, in this case a black and white `hpterm` window, that overrides the specifications for class and client resources.

Syntax and Options

The syntax for `xrdb` is as follows:

```
                ┌-help                                  ┐
                │ { -cpp  path/filename }                │
                │ { -nocpp              }                │
                │-symbols                                │
                │-query                                  │
                │ { -load               }                │
xrdb            │ { -merge              }                │
                │ { -remove             }                │
                │ { -edit  path/filename }               │
                │-backup  string                         │
                │-Dname[= value]                         │
                │-Uname                                  │
                └-Ipath/directory                        ┘
  -display host:display
```

-help — Displays a list of options for `xrdb`.

-display — Specifies the host and display of the server to be loaded with the configuration information.

-query	Displays the current contents of the server's RESOURCE_MANAGER property.
-load	Specifies that **xrdb** should load the file named on the command line into the RESOURCE_ MANAGER property, overwriting the current resources listed there.
-merge	Specifies that **xrdb** should load the file named on the command line into the RESOURCE_ MANAGER property, merging the new resources with the current resources instead of overwriting them.
-remove	Removes the current configuration file from the RESOURCE_MANAGER property.
-edit	Places the contents of the RESOURCE_ MANAGER property into the named file, overwriting resources specified there.
-backup	Specifies a suffix to be appended to the filename used in the **-edit** option to create a backup file.
-cpp	Specifies the path and filename of the C preprocessor to use when loading a configuration file containing **#ifdef** or **#include** statements. **xrdb** works with CPP and other preprocessors as long as they accept the **-D**, **-U**, and **-I** options.
-nocpp	Specifies that **xrdb** should not use a preprocessor before loading the configuration file (the file contains no statements that need preprocessiong).
-symbols	Displays the symbols currently defined for the preprocessor.
-D*name*	Defines a symbol for use with conditional statements in the configuration file used by the RESOURCE_ MANAGER property.
-U*name*	Removes a defined symbol from the RESOURCE_ MANAGER property.
-I*path/directory*	Specifies the search path and directory of **#include** files used in the RESOURCE_ MANAGER.

Examples

The **xrdb** client enables you to swap resource configuration files in and out of the X server's RESOURCE_ MANAGER property. For example, suppose you need to keep switching between your **.Xdefaults** configuration and a special **/projects/proto.defaults** configuration that contains different color resource specifications. To change to **proto.defaults**, type the following:

```
xrdb  -nocpp -load /projects/proto.defaults  Return
```

To see that the resources have actually been swapped, type the following:

```
xrdb  -query  Return
```

Any new clients started now will have the colors specified in **proto.defaults**. To change your existing environment to the **proto.defaults** colors, restart the window manager.

As another example, suppose you are the system administrator of an S300 diskless cluster and that the cluster includes several different types of monitors. One option that **xrdb** gives you is the ability to create a **/usr/lib/X11/syscus.Xdefaults** file containing resource modules headed by **ifdef** statements. One **ifdef** statement for each of the different monitor types. In each user's **.xllstart** script, you replace the existing line calling **xrdb** with the following:

```
xrdb  /usr/lib/X11/syscus.Xdefaults
```

The C language preprocessor reads the class of the monitor (StaticGray, PseudoColor, etc.) passed to it by **xrdb** and, based on that information, select the correct **ifdef** module.

As an elaboration of this, you could include a special module in **syscus.Xdefaults** for novice users (giving them a simplified environment) by placing the following line in their **.x11start** file:

```
xrdb -DUSER=beginner /usr/lib/X11/syscus.Xdefaults
```

This line would select the correct monitor module for the novice's monitor, while tempering the usual resources for that monitor type with resources from the beginner module.

Using National Language Input/Output

Though most character sets are composed of 8-bit characters, some languages (Japanese, Chinese, and Korean) have larger character sets that require 16-bit characters. The X Window System supports the use of 16-bit character input with the National Language Input/Output (NL I/O) subsystem.

To use NL I/O you must have the following:

- The NL I/O subsystem properly installed on your system.
- The appropriate language keyboard *or* an ASCII keyboard.
- The appropriate NL I/O fonts installed in the `/usr/lib/X11/fonts` directory.

Configuring 'hpterm' Windows for NL I/O

You can configure an `hpterm` window to display NL I/O. The process uses the `config keys` and `terminal config` softkeys available with `hpterm` windows to configure the window for NL I/O. Follow the steps outlined in the *Native Language I/O System Administrator's Guide* (HP 92559-90002) and the *Native Language I/O Access User's Guide* (HP 92559-90001).

This configures the `hpterm` client. You must also select the appropriate NL I/O font.

Specifying an NL I/O Font

NL I/O fonts are part of the NL I/O product. You install them in the `/usr/lib/X11/fonts` directory when you install your NL I/O subsystem.

You specify an NL I/O font exactly like you specify any other font. For example, if you want to create an `hpterm` window that uses the Japanese font `jpn.8x18`, use the following command line:

```
hpterm -fn jpn.8x18 & Return
```

Where to Go Next

The next chapter discusses X Window System printing and screen dumping utilities. The chapter after that, chapter 9, discusses the X Window System as an environment for Starbase applications.

8

Printing and Screen Dumps

The X Window System includes clients that enable you to do **screen dumps.** A screen dump is an operation that captures an image from your screen and saves it in a bitmap file. You can then redisplay, edit, or send the file to the printer for hardcopy reproduction.

Read this chapter if you need to "take a picture" of something on the screen for future use or if you want to print what is on your screen.

This chapter discusses the following topics:

- Making a screen dump.
- Displaying a screen dump.
- Printing a screen dump.

Making and Displaying Screen Dumps

X11 windows can be dumped into files by using the **xwd** client. The files can be redisplayed on the screen by using the **xwud** client.

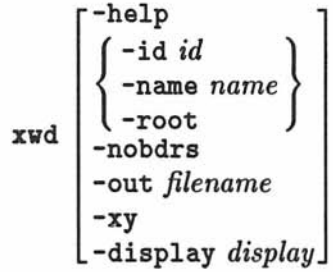
Making a Screen Dump with 'xwd'

The **xwd** client allows you to take a "picture" of a window that is displayed on the screen and store it in a file. The filed picture can then be printed, edited, or redisplayed. You select the window to be dumped either by clicking the mouse on it or by specifying the window name or id on the command line.

The resulting file is called an **xwd**-format bitmap file or an **xwd** screen dump. All of the figures used in this manual, with the exception of the system diagrams in chapter 2, are **xwd** screen dumps.

Syntax and Options

The syntax for **xwd** is as follows:



-help	Provides a brief description of usage and syntax.
-id	Specifies the window to be dumped by its *id* rather than using the mouse to select it.
-name	Specifies the window to be dumped by its *name* rather than using the mouse to select it.
-root	Specifes that the window to be dumped is the root window.
-nobdrs	Dumps the window without borders.
-out	Specifies that the screen dump is to be stored in the file *filename.*
-xy	Selects 'XY' format of storage instead of the default 'Z' format.
-display	Specifies the screen that contains the window to be dumped.

Example 1: Selecting a Window with the Pointer

This example stores a window in a file named **savewindow**, using the pointer to determine which window you want.

1. Display the an hpterm or xterm window.
2. Type:

   xwd -out savewindow Return

   The pointer changes shape, signifying you can select a window to dump.

3. Move the pointer into the window you want to dump. Press and release the select button. The cursor changes back to its normal shape and the window is stored in the file **savewindow**.

Example 2: Selecting a Window with a Name

If you know the name of the window you want to dump, you don't need to use the pointer at all. This example dumps the window named "calendar" to a file named **calendar.dump**.

```
xwd -name calendar -out calendar.dump Return
```

Displaying a Stored Screen Dump with 'xwud'

The **xwd** client allows you to display an **xwd**-format bitmap file on your monitor. You could have created the file earlier with **xwd** or translated it from another format into **xwd** format.

Note

The image to be restored has to match the depth of the system on which it is to be restored. For example, an image created and stored using a depth of four cannot be restored on a system with a different depth.

Syntax and Options

The syntax for **xwud** is as follows:

```
xwud [-help
      -in filename
      -inverse
      -display host:display.screen]
```

-help	Displays a brief description of the options.
-in	Specifies the file containing the screen dump.
-inverse	Reverses black and white from the original monochrome dump.
-display	Specifies the screen on which to display the dump.

Example

This example displays the bitmap file **myfile**.

```
xwud -in myfile Return
```

Printing Screen Dumps

Before you can print the screen dump, you need to ensure that your printer is connected and talking to your computer.

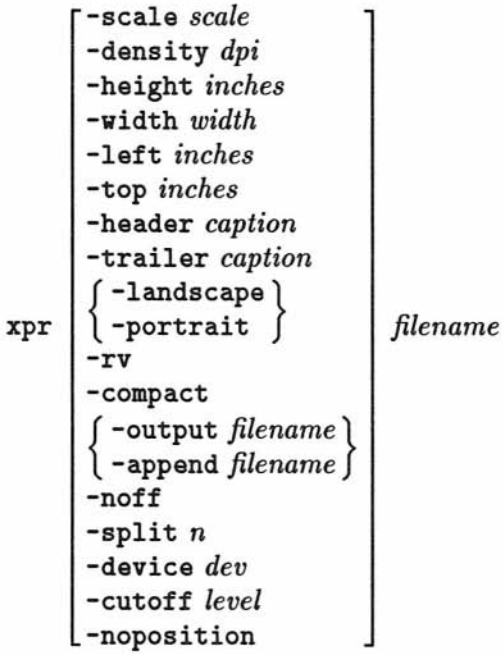
If you are the system administrator, refer to the *HP-UX System Administrator Manual* for information about these tasks. If you're not the system administrator, ask the person who is to perform these tasks:

- Connect the printer to your computer.
- Create a device file for the printer on your computer.
- Run the print spooler.

Printing Screen Dumps with 'xpr'

xpr prints a screen dump that has been produced by **xwd**.

Syntax and Options



-scale	Specifies a multiplier for pixel expansion. The default is the largest that will allow the entire image to fit on the page.
-density	Specifies the dots per inch for the printer.
-height	Specifies the maximum height in inches of the window on the page.
-width	Specifies the maximum width in inches of the window on the page.
-left	Specifies the left margin in inches. The default is centered.
-top	Specifies the top margin in inches. The default is centered.
-header	Specifies a caption to print above the window.
-trailer	Specifies a caption to print below the window.
-landscape	Prints the window in landscape mode. The default prints the long side of the window on the long side of the paper.

-portrait	Prints the window in portrait mode. The default prints the long side of the window on the long side of the paper.
-rv	Reverses black and white from the original screen.
-compact	Provides efficient printer directions for a window with lots of white space (Postscript printers only).
-output	Specifies a file to store the output in.
-append	Adds the window to the end of an existing file.
-noff	Specifies that the window should appear on the same page as the previous window. Used with -append.
-split	Prints the window on *n* pages. Not applicable to HP printers.
-device	Specifies the printer to use.
-cutoff	Specifies intensity for converting color to monochrome for printing on a LaserJet printer.
-noposition	Bypasses header positioning, trailer positioning, and image positioning commands for the LaserJet and PaintJet printers.
filename	Specifies the bitmap file to print.

Example

Suppose you want to print a bitmap file named **myfile** that you previously created with **xwd**. You want to print the file on a LaserJet printer in portrait mode with black and white the reverse of the original bitmap file.

You must first determine the device type for your system. The example uses **lj**, but your system could have a different device type assigned. Follow these steps to find the device type for your system:

1. Type the following:

```
lpstat -t Return
```

2. In response, you should see something like this:

```
system default destination: lj
device for lj:  /dev/lj2000
device for lj2: /dev/lj2
lj accepting requests since Jan 05 11:37
lj2 accepting requests since Jan 05 12:11
printer lj is now idle.  enabled since Jan 05 11:37
printer lj2 is now idle.  enabled since Jan 05 12:11
```

3. From the above example of `lpstat -t`, you can see that there are two printer device types for this system, `lj` and `lj2`. Use the device type for your system in place of `lj` in the example below.

```
xpr -device lj -portrait -rv myfile | lp -oraw Return
```

Reversing colors is often used when preparing illustrations for documents. The original illustration can be done in white with a black background, which is easy to see on computer displays, but reversed to give a black drawing on a white background, which is common in printed material.

Moving and Resizing the Image on the Paper

You may not always want to have the image print exactly in the same size or location as the default choices place it.

Sizing Options

The three sizing options for `xpr` are:

-scale	Each bit of the bitmap is translated into a grid of the size you specify. For example, if you specify a scale of 5, each bit in the bitmap is translated into a 5 by 5 grid. This is an easy way to increase the size without refiguring the height and width.
-height	The maximum height in inches of the image on the page.
-width	The maximum width in inches of the image on the page.

The actual printed size could be smaller than ((-height)) and ((-width)) if other options, such as the orientation ones, conflict with them.

Location Options

The two location options for **xpr** are:

-left The left margin in inches.

-top The top margin in inches.

If ((-left)) is not specified, the image is centered left-to-right. If ((-top)) is not specified, the image is centered top-to-bottom.

Orientation Options

The two orientation options to **xpr** are:

-landscape The image is printed so that the top of the image is on the long side of the paper.

-portrait The image is printed so that the top of the image is on the short side of the paper.

If neither option is specified, **xpr** will position the image so that the long side of the image is on the long side of the paper. However, you can force it to print either in landscape mode or portrait mode by using the appropriate option.

Unless told otherwise by the sizing options, **xpr** makes the image as big as can fit in the orientation specified.

Printing Multiple Images on One Page

xpr normally prints each image on a separate page. The -noff option is used to print more than one image on a page.

Printing Color Images

Printing Color Images on a PaintJet

Use the device name **pjet** to direct output to a PaintJet.

If you have a PaintJet on your system, there may be a different device type. Use the `lpstat -t` command to determine the device types in use for your system. Ask your System Administrator for assistance if you are still unsure of your system's device type.

For example, the following command prints a bitmap file named `myfile` on a PaintJet.

```
xpr -device pjet myfile Return
```

Printing Color Images on a LaserJet

Color images printed on a LaserJet will be in black and white instead of color. Often you need do nothing but specify `lj` as the device. If your original color image contained many colors of the same intensity, the LaserJet version may be all light or all dark. If that happens, use the ((-cutoff)) option to change the mapping of color intensities. Anything above the cutoff value is white and anything below is black. Note that the default cutoff value is 50 percent.

Where To Go Next

Only one chapter left! Chapter 9 covers using the powerful Starbase graphics library from X11.

9

Using Starbase on X11

Starbase is a powerful graphics library from Hewlett-Packard. It provides two-dimensional and three-dimensional graphics, a variety of input and output capabilities, and high performance features, such as hidden surface removal, shading, and light sourcing.

This chapter describes how X11 interacts with Starbase. It does not describe Starbase itself. For detailed information about Starbase features, refer to *Starbase Programming with X11* in the Starbase documentation.

This chapter covers the following topics:

- Using the **X*screens** file to control display options.
- Starting the X11 server.
- Opening and destroying windows for Starbase applications.
- Creating transparent windows.
- Conversion utilities.

Using the X*screens File

This section reviews some concepts that you need to understand before starting the Starbase server:

- **X*screens** file.
- Monitor Type.
- Operating modes.
- Double buffering.

The `X*screens` file is a system file that contains the screen configurations you want to use. Before you run a Starbase application, you should ensure that the configuration is correct for Starbase. The `X*screens` file is described in chapter 7.

The following sections describe options you should be aware of when running Starbase on X11. It also explains how to specify those options in the `X*screens` file. If the `X*screens` file you are using doesn't have the correct entries to do what you want, edit it to include the correct information.

Generally, once you have modified the `X*screens` file, you won't have to change it again unless you add a new monitor.

Monitor Type

Starbase can run on a wide variety of graphics monitors. More sophisticated monitors provide a wider choice of options. The following table shows which options are available for different monitors.

Table 9-1. Display Hardware and Available Options

With this display hardware ...	You can use these options ...			
HP Part Number	**Maximum Planes (colors)**	**Modes**	**Double buffer**	**Depth**
HP 98542A	1 Image (monochrome)	Image		
HP 98543A	4 Image (monochrome)	Image	√	
HP 98544B	1 Image (monochrome)	Image		
HP 98545A	4 Image (monochrome)	Image		
HP 98547A	6 Image (64)	Image	√	
HP 98548A	1 Image (monochrome)	Image		
HP 98549A	6 Image (64)	Image	√	
HP 98550A	2 Overlay (4) 8 Image (256)	Image Overlay	√	
HP 98720A	3 Overlay (8) 8-24 Image (256 from 16 million)	Image Overlay Stacked	√ √	√ √
HP 98730A	4 Overlay 8-24 Image (256 from 16 million)	Image Overlay Stacked Combined	√ √ √	√ √ √

Operating Modes

Image and Overlay Planes

Monitors can have two kinds of display planes, image and overlay. The image plane allows the monitor hardware to help the graphics commands run faster and more efficiently.

Server Operating Modes

The operating mode results from the way you specify the image and overlay screens in the `X*screens` file.

The four different modes are:

Overlay mode	The X11 server operates only in the overlay planes. Starbase can display in its "raw" mode, writing directly to the image planes, rather than to a window. A "transparent" overlay window can look through to the Starbase display in the image planes. The Starbase double buffering feature does not apply in this mode.
Image mode	This is the only mode available on those displays that do not have overlay planes. Even if overlay planes are available, you may want to use image mode to have a greater number of colors available.
Stacked screen mode	In this mode, the image planes are treated as one screen and the overlay planes as another, separate screen, providing twice as much screen space. The pointer is moved to the edge of the display to switch between the overlay and image planes.
Combined mode	This mode (available only on the HP 98730A) treats the overlay and image planes as a single device that provides multiple window types to client programs.

Monochrome monitors and low-level color monitors run in the image mode.

Documentation for the Starbase application program will tell you which mode or which plane the application expects.

The following examples show how `X*screens` entries vary for each mode.

Example 1: Image Mode

This example shows an image mode entry in the `X*screens` file. The same entry is used regardless of the type of monitor. The number of colors available to you depends on the monitor. The entry may also have the options discussed in the next few sections.

```
/dev/crt
```

Example 2: Overlay Mode

This example shows an overlay mode entry in the `X*screens` file for monitors that support overlay mode. The number of colors you are able to use depends on your display adaptor.

```
/dev/ocrt
```

Example 3: Stacked Mode

This example shows the entries for stacked mode for monitors that are able to support stacked mode. Stacked mode is indicated by having each entry on a separate line. Image plane entries can have the options discussed in the next few sections. Note that the order of the entries determines the order of the screens. Screen 0 is the first entry, screen 1 is the second entry, and so on.

```
/dev/ocrt
/dev/crt
```

Example 4: Combined Mode

This example shows how a combined mode entry is made in the `X*screens` file for monitors that support combined mode. Combined mode is indicated by having the entries for the overlay and image planes on the same line, overlay planes first. Image plane entries can have the options discussed in the next few sections.

```
/dev/ocrt /dev/crt
```

Double Buffering

This feature does not apply to monochrome monitors or when the X11 server is running in the overlay planes.

Double buffering means that Starbase uses half of the color planes of your monitor to display to the screen, and uses the other half to compute and draw the next screen display. This provides smooth motion for animation, and it is also faster. However, double buffering reduces the number of colors available for displaying on the screen at one time. Some applications require double buffering. If you run a double-buffered application in single buffered mode, the display will flash or flicker rapidly.

Example 1: Image Mode

```
/dev/crt doublebuffer
```

Example 2: Stacked Mode

```
/dev/ocrt
/dev/crt doublebuffer
```

Example 3: Combined Mode

```
/dev/ocrt /dev/crt doublebuffer
```

Screen Depth

You can specify a screen depth for image planes in the `X*screens` file. Valid depths for regular (single buffer) mode are 8 and 24. Valid depths for doublebuffer mode are 8, 16, and 24. The depth of overlay planes is determined by the `/dev` entry in `X*screens`. The depth for the HP 98550A is 2 overlay planes: the depth for the HP 98720A has 3 overlay planes; and the depth for the HP 98730 can be either 3 or 4 overlay planes.

More planes means more colors can be displayed simultaneously. For computer-generated graphics to look as realistic as photographs, thousands of colors must be shown at the same time. 8 planes means that 2^8 (256) colors can be shown, while 24 planes means that 2^{24} (16 million) colors can be shown. Note that depth is specified only when you have more than one depth available. This feature is feature is available only on the HP 98720A and HP 98730A Display Controller.

Example 1: Image Mode

The following example shows an `X*screens` file entry for an HP 98720A monitor running in image mode. Windows can have 8 planes (256 colors) displayed simultaneously.

```
/dev/crt depth 8
```

Example 2: Combined Mode

The following example provides two doublebuffered depths in the image planes: depth 8 (16 planes/2) and depth 12 (24 planes/2). That is, some windows in

the image planes could have a depth of 8 planes, while others could have a depth of 12 planes. This is possible only in combined mode.

```
/dev/ocrt /dev/crt depth 16 depth 24 doublebuffer
```

Starting the X11 Server

Once you have ensured that the options you need are in the **X*screens** file, type

```
x11start -- :n Return
```

where *n* is replaced by the number of the **X*screens** file you want to have in effect. For example, if you have all your display options in the **X0screens** file, type **x11start -- :0** to start the server.

Window-Smart and Window-Naive Programs

Window-smart applications are able to create and destroy the windows in which they operate.

Window-naive (sometimes called window-dumb) applications aren't able to create and destroy windows on their own. They need help from the X11 system.

Although this chapter discusses window-smart and window-naive applications in relation to Starbase, the same procedures are used to start non-Starbase programs.

Is My Application Window-Smart or Window-Naive?

If you are using an existing application, the documentation that comes with the application will tell you how to start it. You don't have to worry whether it is window-smart or window-naive, just follow the directions.

If you are writing a new application using Starbase, use the **xwcreate** and **xwdestroy** commands. Rather than typing the commands each time you want to test the new program, put the commands in a file, then execute the file to start the application. In this case, the application is window-naive but the file is window-smart.

Running Window-Smart Programs

From an **hpterm** window, type the name of the Starbase program you want to run.

For example, the following command will start a hypothetical Starbase application named **planetarium** that displays a moving display of the night sky. Assume that the program is in the /users/funstuff/ directory on your computer.

```
/users/funstuff/planetarium Return
```

Running Window-naive Programs

Window-naive programs cannot open and close the window they need to run in, so you must do it for them with clients (a terminal emulator, for example). Programs that use the Starbase graphics library are window-naive.

Most window-naive programs are able to run in the X Window System environment using the **sox11** device driver. The **sox11** driver is described in the *Starbase Device Drivers* manual. But window-naive clients still need help to create and destroy the windows they display their output in.

To enable such window-naive graphics programs to run within X, you need four special helper clients to create and destroy the windows used by the naive graphics programs. The clients are:

- **gwind**
- **xwcreate**
- **xwdestroy**
- **gwindstop**

gwind runs in the background and services requests from the other three helper clients. When requested by **xwcreate**, **gwind** creates a window in which

an application can display its output; when requested by **xwdestroy**, **gwind** destroys the window. *You don't need to start the* **gwind** *program,* **xwcreate** *and* **xwdestroy** *start it for you.*

The next sections cover:

- Creating a window
- Destroying a window

An example showing all of these steps follows the discussion.

Creating a Window with 'xwcreate'

xwcreate requests **gwind** to create a window for a window-naive graphics program to use for its output. The graphics program must exist on the same computer that is running **xwcreate**. If **gwind** is not already running when **xwcreate** is executed, **xwcreate** will start **gwind**. Once **xwcreate** has created a window, you can use the window to run your graphics program. When you finish that application, you can use the same window to run another graphics program if you wish.

When to Use 'xwcreate'

Use **xwcreate** from the command line.

Syntax and Options

xwcreate [**-display** *host:display.screen* | **-parent** *parent* | **-geometry** *width*×*height*±*col*±*row* | **-r** | **-bg** *color* | **-bw** *pixels* | **-bd** *color* | **-depth** *depth* | **-wmdr** *directory* | **-title** *name*]

-display Specifies the screen the window will appear on

-parent Names a window to be the parent of the window being created.

-geometry	Specifies desired size and location of window.
-r	Specifies backing store. Default is no backing store.
-bg	Specifies the background color. The default is black.
-bw	Specifies the border width in pixels. The default is 3 pixels wide.
-bd	Specifies the border color. The default is white.
-depth	Specifies the depth of the window. The default is the same depth as its parent.
-wmdir	Specifies the name of the directory containing the pty file for the window.
-title	Specifies the name the window will be called.

The **depth** option is where you tell the window manager what set of planes you want the window to be in. If you specify nothing, the window will be created in the overlay planes. If you specify a depth, the window will be placed in the image plane with the depth (number of color planes) you specify.

The following example creates a window named "foo:"

```
xwcreate -title foo Return
```

Destroying a Window with 'xwdestroy'

xwdestroy destroys the window created by **xwcreate**. If that window is the only graphics window present at that time, **gwind** will also be terminated.

When to Use 'xwdestroy'

Use **xwdestroy** from the command line.

Syntax and Options

xwdestroy [−**wmdir** *path/directory*] *window1 window2 ...*

-wmdir	Specifies the directory containing the pty file for the window.

window Specifies the window or windows to be destroyed.

The following example will destroy a window named "foo:"

```
xwdestroy foo
```

Destroying a Window with 'gwindstop'

`gwindstop` destroys all windows created by `gwind` in the specified directory. If, however, you use `xwdestroy` to remove the *last* window opened for graphics use, `xwdestroy` will remove the other windows as well. You *do not* need to use `gwindstop`.

Caution

You must use `xwdestroy` or `gwindstop` to get rid of a window after you have finished running your graphics application. Do *not* use `kill` to remove the `gwind` process associated with the window. If you should accidentally do so, you must type the command `rm $WMDIR/wm`. Failure to do this will result in `xwcreate` not running the next time you call it.

When to Use 'gwindstop'

Use `gwindstop` from the command line.

Syntax and Options

```
gwindstop [directory] [directory] ...
```

directory The directory containing the pty files for the windows to be destroyed.

Running Starbase in Raw Mode

If your monitor doesn't support overlay planes, you can run Starbase in "raw" mode, which means that Starbase writes to the entire screen rather than to a window. You then use a transparent window to see through to the Starbase output.

For information about Starbase raw mode, refer to the Starbase documentation.

Using Transparent Windows

Transparent windows allow you to look through an overlay window into the image planes.

Creating a Transparent Window with 'xseethru'

xseethru is a transparent overlay-plane window used to see through the overlay planes to the image planes. It is used in stacked or overlay mode.

When to Use 'xseethru'

Use **xseethru** from the command line.

Syntax and Options

xseethru [**-geometry** *width*×*height*±*col*±*row*
-display *host:display.screen*]

-geometry — The geometry used to create the window. Refer to chapter 4 for more information about geometry.

-display — The screen the window will appear on. Refer to chapter 4 for more information about display.

Example

This example opens a transparent window 100-pixels by 100-pixels in size and located 50 pixels from the left and 25 pixels from the top of the screen.

```
xseethru -geometry 100x100+50+25 Return
```

Creating a Transparent Window with 'xsetroot'

xsetroot allows you make the root window transparent when you are running X in the overlay planes.

When to Use 'xsetroot'

Use **xsetroot** from the command line.

Syntax and Options

xsetroot [−**solid** *color*]

-solid Sets the window color to *color*.

Example

This example turns the root window into a transparent window.

```
xsetroot -solid transparent Return
```

Creating a Transparent Background Color

Any window may have **transparent** as its background color.

Chapter 4 explains how to set colors in a window. This example opens an **hpterm** window with a transparent background color.

```
hpterm -bg transparent Return
```

Conversion Utilities

This section shows you how to use the utilities **sb2xwd** and **xwd2sb**.

Converting Starbase Format to 'xwd' Format using 'sb2xwd'

sb2xwd converts Starbase format window files into **xwd** format pixmaps. The pixmaps can then be printed by using **xpr** or displayed on the screen by using **xwud**, both of which are described in chapter 8.

When to Use 'sb2xwd'

Use **sb2xwd** from the command line.

Syntax and Options

sb2xwd < *filename* > *filename*

<*filename* The Starbase window file to be converted.

>*filename* The **xwd** pixmap file name.

Example

This example translates the Starbase window file named **mystar** into an **xwd** pixmap file named **myxwd**, then prints it on an HP LaserJet printer.

```
sb2xwd < mystar > myxwd
xpr -dev lj myxwd | lp -oraw
```

Converting 'xwd' Format to StarBase Format using 'xwd2sb'

xwd2sb is the opposite of **sb2xwd**. It converts **xwd** format pixmaps into Starbase format window files. An example of this would be creating graphics in Starbase, converting to **xwd**, editing the graphic, and then converting back to Starbase.

The **file_to_bitmap** Starbase command can then be used to load the converted file into a Starbase window file.

When to Use 'xwd2sb'

Use `xwd2sb` from the command line.

Syntax and Options

`xwd2sb <` *filename* `>`*filename*

`<`*filename*	The `xwd` bitmap file to be converted.
`>`*filename*	The Starbase window file filename.

Example

This example dumps an `xwd` window named `sample` into a bitmap file called `myxwd`, translates it into a Starbase window file, and stores it in a file called `mystar`.

```
xwd sample -name myxwd
xwd2sb < myxwd > mystar
```

The file `mystar` is now in a format to be used directly by the Starbase commands.

A

Reference Information

This section contains reference information about clients included with the X Window System and about the X protocol and server itself. The entries are arranged alphabetically, each starting on its own "page 1."

MAN Pages

bdf(4)	xload(1)
bitmap(1)	xmodmap(1)
gwindstop(1)	xpr(1)
hpterm(1)	xprkbd(1) on xmodmap(1) MAN page
hpwm(1)	xrdb(1)
resize(1)	xrefresh(1)
rgb(1)	xseethru(1)
sb2xwd(1)	Xserver(1)
uwm(1)	xset(1)
X(1)	xsetroot(1)
x11start(1)	xterm(1)
xclock(1)	xwcreate(1)
xfc(1)	xwd(1)
xfd(1)	xwd2sb(1)
xhost(1)	xwdestroy(1)
xinit(1)	xwininfo(1)
xinitcolormap(1)	xwud(1)

NAME

bdf - Bitmap Distribution Format 2.1

DESCRIPTION

A "bdf" file is a file conforming to Bitmap Distribution Format 2.1. It is used for specifying fonts for the X11 Windowing System. In bdf format, the font is transportable between systems and is converted to a server natural format via the use of **xfc**(see xfc(1) for more details).

FILE FORMAT

STARTFONT 2.1
This must be the first line of the file.

COMMENT
This must be the second line in the file and can be 1 or more lines. These lines are ignored by the font compiler.

FONT *family_name-face_name*
This must be the third line in the file. Examples of a family name are "oldenglish", "6x10.bits", "germanic". Examples of face name are "bold" and "italic".

SIZE *point_size x_resolution y_resolution*
This must be the fourth line in the file. *point_size* is usually based on 72 points per inch. If a font has a point size of 8, it will be 1/9th of an inch high; point size of 16 will be 1/4 of an inch high. The *x_resolution* and *y_resolution* is the pixels per inch on the display for which the font was created.

FONTBOUNDINGBOX *width height x_displacement y_displacement*
This must be the fifth line in the file. *width* and *height* are the maximum width and height in pixels of the font. *x_displacement* and *y_displacement* indicate (again in pixels) the lower left corner of the font with respect to the origin.

STARTPROPERTIES *p*
This is an optional keyword. If included, *p* indicates the number of special properties following it. See "SPECIAL PROPERTIES" for a complete description of the special properties available.

ENDPROPERTIES
If there is a **STARTPROPERTIES**, there must be an **ENDPROPERTIES**.

CHARS *c*
This keyword must be the next statement after **FONTBOUNDINGBOX** (or **ENDPROPERTIES** if special properties are declared). *c* indicates the number of characters the font will contain. **CHARS** is followed by *c* **STARTCHAR**, **ENDCHAR** pairs. See "CHARACTER DEFINITION" for details on the information supplied by the **STARTCHAR** and **ENDCHAR** pair.

ENDFONT
This must be the last line in the file. It indicates the end of the information the font compiler is to process.

CHARACTER DEFINITION

STARTCHAR *name*
This is the first keyword for each character to be described for the font. *name* is the descriptive name of the glyph, i.e. "j", "x", "1", etc.

ENCODING *integer*
This must be the first keyword to follow **STARTCHAR**. It indicates the Adobe Standard Encoding value for *name*. *integer* must be a positive integer. If it is a -1, it is assumed that the character is not a member of the Adobe Standard Encoding. In this case, the -1 can be followed by an optional integer specifying the glyph index.

SWIDTH *scalable_width_x scalable_width_y*
This is the scalable width of the character in x and y. To calculate the the width in device pixels from the scalable width, multiply **SWIDTH** by *p* times *r* divided by 72000 where *p* is the size of the character in points and *r* is the device resolution in pixels per inch.

DWIDTH *width_x width_y*
width_x and *width_y* are a vector in device units indicating the position of the next character's origin relative to this character.

BBX *BBw BBh BBox BBoy*
BBw and *BBh* are the width and height of this character. *BBox* and *BBoy* are the x and y displacement from the origin to the lower left corner of the character.

BITMAP
This keyword appears on the line by itself and is followed by *BBh* lines of hex-coded bitmap, padded on the right with zeros to the nearest multiple of 8.

ENDCHAR
For each **STARTCHAR**, there must be an **ENDCHAR**.

SPECIAL PROPERTIES
The special properties information is saved in the special properties list of the font for access by any application wishing more information about the font. The currently supported special properties are:

FONT_ASCENT *integer*
Determines the top boundary of the font.

FONT_DESCENT *integer*
Determines the bottom boundary of the font.

DEFAULT_CHAR *integer*
This is the default character to use for all undefined characters.

POINT_SIZE *integer*
Same as *point_size* specified in **SIZE**.

FAMILY_NAME *string*
Same as *family_name* specified in **FONT**.

RESOLUTION *integer*
The resolution in pixels per inch of the display for which the font was created.

X_HEIGHT *integer*
This specifies the height of the lower case "x" in quarter-dot units (four quarter-dot units equals one pixel).

WEIGHT *integer*
Specifies the thickness of the strokes used in designing the font. The range is -7 to 7 (thin - thick).

ORIGIN
Character Bitmap Distribution Format 2.1, Adobe Systems, Inc.

SEE ALSO
X(1), Xserver(1), xfc(1)

Series 300 and 800 Only

NAME

bitmap - bitmap editor for X

SYNOPSIS

bitmap [options] *filename* [WIDTHxHEIGHT]

DESCRIPTION

bitmap lets you interactively create bitmaps, or edit previously created bitmaps. A bitmap is simply a rectangular array of 0 and 1 bits. The X Window System uses bitmaps in defining clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

When you run *bitmap,* you are given a magnified version of the bitmap, in which each bit is shown as a large square, as if it were a piece of graph paper. The pointer can be used to set, clear, or invert individual squares, and to invoke commands to set, clear or invert larger rectangular areas of the bitmap. Other commands may be used to move or copy rectangular areas from one part of the bitmap to another, and to define a 'hot spot' (a special single point on the bitmap, which is useful when the bitmap is used as an X cursor).

The output of the *bitmap* program is a small C code fragment. By #include'ing such a program fragment in your program, you can easily declare the size and contents of cursors, icons, and other bitmaps that your program creates to deal with the X Window System.

OPTIONS

This program accepts the following options:

-help This option (or any other unsupported option) will cause a brief description of the allowable options and parameters to be printed.

-display *display*
This option specifies the server to be used. See *X(1)* for details.

-geometry *geometry*
This option specifies the placement and size of the bitmap window on the screen. See *X(1)* for details.

-nodashed
This option indicates that the grid lines in the work area should not be drawn using dashed lines. Although dashed lines are prettier than solid lines, on some servers they are significantly slower.

-bw *number*
This option specifies the border width in pixels of the main window.

-fn *font* This option specifies the font to be used in the buttons.

-fg *color*
This option specifies the color to be used for the foreground.

-bg *color*
This option specifies the color to be used for the background.

-hl *color*
This option specifies the color to be used for highlighting.

-bd *color*
This option specifies the color to be used for the window border.

-ms *color*
This option specifies the color to be used for the pointer (mouse).

When *bitmap* starts, it first tries to read the specified file (see FILE FORMAT). If the file already exists, it creates a window containing a grid of the appropriate dimensions.

If the file does not exist, *bitmap* will create a window for a bitmap of the size specified by *WIDTHxHEIGHT* (e.g. 7x9, 13x21). The bitmap will start out empty. If *WIDTHxHEIGHT* is not specified either on the command line or in the "Dimensions" X Default, 16x16 will be assumed.

The window that *bitmap* creates has four parts. The largest section is the checkerboard grid, which is a magnified version of the bitmap you are editing. At the upper right is a set of commands that you can invoke with any pointer button. Below the commands is an "actual size" picture of the bitmap you are editing; below that is an inverted version of the same bitmap. Each time you alter the image in the grid, the change will be reflected in the actual-size versions of the bitmap.

If you use a window manager to make the *bitmap* window larger or smaller, the grid squares will automatically get larger or smaller as well.

COMMANDS

(Note for users of color displays: In all of the following, "white" means the background color, and "black" means the foreground color.)

When the cursor is in the checkerboard region, each pointer button has a different effect upon the single square that the cursor is over:

Button 1 (usually the left button) sets the indicated square.

Button 2 (usually the middle button) inverts the indicated square.

Button 3 (usually the right button) clear the indicated square.

The various commands are invoked by pressing any pointer button in the corresponding command box:

Clear All
clears all squares in the bitmap. This is irreversible, so invoke it with care.

Set All sets all squares in the bitmap. This is irreversible, so invoke it with care.

Invert All
inverts all squares in the bitmap.

Clear Area
clears a rectangular area of the bitmap. After you click over this command, the cursor turns into an 'upper-left corner'. Press any pointer button over the upper-left corner of the area you want to clear, *hold the button down* while moving the pointer to the lower-right corner of the area you want to clear, and then let the button up.

While you are holding down the button, the selected area will be covered with X's, and the cursor will change to a 'lower-right corner'. If you now wish to abort the command without clearing an area, either press another pointer button, move the cursor outside the grid, or move the cursor to the left of or above the upper-left corner.

Set Area sets a rectangular area of the bitmap. It works the same way as the *Clear Area* command.

Invert Area
inverts a rectangular area of the bitmap. It works the same way as the *Clear Area* command.

Copy Area
copies a rectangular area from one part of the grid to another. First, you select the

rectangle to be copied, in the manner described above under *Clear Area* above. Then, the cursor will change to an "upper-left corner". When you press a pointer button, a destination rectangle will overlay the grid; moving the pointer while holding down the button will move this destination rectangle. The copy will occur when you let up the button. To cancel the copy, move the pointer outside the grid and then let up the button.

Move Area
works identically to *Copy Area, except* that it clears the source rectangle after copying to the destination.

Overlay Area
works identically to *Copy Area*, except that it does a binary OR of the source rectangle with the destination.

Line will draw a line between two points.

Circle will draw a circle given the center and a radius

Filled Circle
will draw a filled circle given the center and radius of the circle.

Flood Fill
will fill a bounded area. Note: if the area is not completely bounded, you will get a 'leak' and the entire grid will be filled.

Set HotSpot
designates a point on the bitmap as the "hot spot". If a program is using your bitmap as a cursor, the hot spot indicates which point on the bitmap is the "actual" location of the cursor. For instance, if your cursor is an arrow, the hot spot should be the tip of the arrow; if your cursor is a cross, the hot spot should be where the perpendicular lines intersect.

Clear HotSpot
removes any hot spot that was defined on this bitmap.

Write Output
writes the current bitmap value to the file specified in the original command line. If the file already exists, the original file is first renamed to **filename~** (in the manner of *emacs(1)* and other text editors).

If either the renaming or the writing cause an error (e.g. "Permission denied'), a dialog window will appear, asking if you want to write the file */tmp/filename* instead. If you say yes, all future "Write Output" commands will write to */tmp/filename* as well. See below for the format of the output file.

Quit exits the *bitmap* program. If you have edited the bitmap and have not invoked *Write Output,* or you have edited it since the last time you invoked *Write Output,* a dialog window will appear, asking if you want to save changes before quitting. "Yes" does a "Write Output" before exiting; "No" just exits, losing the edits; "Cancel" means you decided not to quit after all.

FILE FORMAT

Bitmap reads and writes files in the following format, which is suitable for #include'ing in a C

program:
#define *name*_width 9
#define *name*_height 13
#define *name*_x_hot 4
#define *name*_y_hot 6
static char *name*_bits[] = {
0x10, 0x00, 0x38, 0x00, 0x7c, 0x00, 0x10, 0x00, 0x10, 0x00, 0x10, 0x00,
0xff, 0x01, 0x10, 0x00, 0x10, 0x00, 0x10, 0x00, 0x7c, 0x00, 0x38, 0x00,
0x10, 0x00};

The variables ending with *_x_hot* and *_y_hot* are optional; they will be present only if a hot spot has been defined for this bitmap. The other variables must be present.

The *name* portion of the five variables will be derived from the name of the file that you specified on the original command line by
(1) deleting the directory path (all characters up to and including the last '/', if one is present)
(2) deleting the extension (the first '.', if one is present, and all characters beyond it)

For example, invoking *bitmap* with filename */usr/include/bitmaps/cross.bitmap* will produce a file with variable names *cross_width, cross_height,* and *cross_bits* (and *cross_x_hot* and *cross_y_hot* if a hot spot is defined).

It's easy to define a bitmap or cursor in an X program by simply #include'ing a bitmap file and referring to its variables. For instance, to use a cursor defined in the files *this.cursor* and *this_mask.cursor,* one simply writes

```
#include "this.cursor"
#include "this_mask.cursor"
Pixmap source = XCreateBitmapFromData (display, drawable, this_bits, this_width, this_height);
Pixmap mask = XCreateBitmapFromData (display, drawable, this_mask_bits,
  this_mask_width, this_mask_height);
Cursor cursor = XCreatePixmapCursor (display, source, mask, foreground, background,
  this_x_hot, this_y_hot);
```

where *foreground* and *background* are XColor values.

An X program can also read a bitmap file at runtime by using the function *XReadBitmapFile.*

The bits are in XYBitmap format, with bitmap_unit = bitmap_pad = 8, and byte_order = bitmap_bit_order = LSBFirst (least significant bit and byte are leftmost).

For backward compatibility with X10, *bitmap* can also read in a file where the "bits" array is declared as "static short foo_bits[]" and consists of an array of 16-bit hex constants. This is interpreted as a XYBitmap with bitmap_unit = bitmap_pad = 16, byte_order = bitmap_bit_order = LSBFirst. If you modify the bitmap after reading in such a file, *bitmap* will always write the file back out in standard X11 format.

X DEFAULTS

The *bitmap* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

Background
The window's background color. Bits which are 0 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is "white".

BorderColor
The border color. This option is useful only on color displays. The default value is "black".

BorderWidth
The border width. The default value is 2.

BodyFont
The text font. The default value is "variable".

Foreground
The foreground color. Bits which are 1 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is "black".

Highlight
The highlight color. *bitmap* uses this color to show the hot spot and to indicate rectangular areas that will be affected by the *Move Area, Copy Area, Set Area,* and *Invert Area* commands. If a highlight color is not given, then *bitmap* will highlight by inverting. This option is useful only on color displays.

Mouse The pointer (mouse) cursor's color. This option is useful only on color displays. The default value is "black".

Geometry
The size and location of the bitmap window.

Dimensions
The *WIDTHxHEIGHT* to use when creating a new bitmap.

ENVIRONMENT

DISPLAY - the default host and display number.
XENVIRONMENT - the name of the defaults file to use.

DIAGNOSTICS

The following messages may be printed to the standard error output. Any of these conditions aborts *bitmap* before it can create its window.

"bitmap: could not connect to X server on *host:display*"

Either the display given on the command line or the DISPLAY environment variable has an invalid host name or display number, or the host is down, or the host is unreachable, or the host is not running an X server, or the host is refusing connections.

"bitmap: no file name specified"

You invoked *bitmap* with no command line arguments. You must give a file name as the first argument.

"bitmap: could not open file *filename* for reading -- *message*"

The specified file exists but cannot be read, for the reason given in <message> (e.g., permission denied).

"bitmap: invalid dimensions *string*"
"bitmap: dimensions must be positive"

The second command line argument was not a valid dimension specification.

"bitmap: Bitmap file invalid"

The input file is not in the correct format; the program gave up when trying to read the specified data.

The following messages may be printed after *bitmap* creates its window:

"bitmap: Unrecognized variable *name* in file *filename*"

bitmap encountered a variable ending in something other than *_x_hot*, *_y_hot*, *_width*, or *_height* while parsing the input file. It will ignore this variable and continue parsing the file.

"bitmap: XError: *message*"
"bitmap: XIOError"

A protocol error occurred. Something is wrong with either the X server or the X library which the program was compiled with. Possibly they are incompatible. If the server is not on the local host, maybe the connection broke.

NOTES

The old command line arguments aren't consistent with other X programs.

If you move the pointer too fast while holding a pointer button down, some squares may be 'missed'. This is caused by limitations in how frequently the X server can sample the pointer location.

There is no way to write to a file other than the one specified on the command line.

There is no way to change the size of the bitmap once the program has started.

There is no "undo" command.

If you read in an X10-format bitmap, the "Quit" and "Write Output" commands won't write out a new, X11-format, file unless you've changed at least one square on the bitmap. You can work around this by simply inverting a square and then inverting it back again.

This program would make a wonderful X toolkit application.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1), *Xlib - C Language X Interface*, particularly the section "Manipulating Bitmaps".

Series 300 and 800 Only

NAME

gwindstop - terminate the window helper facility

SYNOPSIS

gwindstop [directory] [directory] ..

DESCRIPTION

gwindstop

destroys windows and their associated pty files from named directories. The windows must have been created earlier by xwcreate(1).

directory

is the name of the directory where the pty files for the windows reside. If **directory** name is not supplied, */dev/screen* is taken to be the desired directory. Otherwise, if the directory argument implies an absolute pathname, then it will be taken to be the desired directory. Otherwise, the directory name will be taken to be relative to the value of the environment variable $WMDIR. If $WMDIR is not defined in the environment, the directory name will be taken to be relative to */dev/screen*. Note: if $WMDIR is defined in the environment, it must represent an absolute pathname.

DIAGNOSTICS

If the windows in the indicated directory are successfully destroyed, then the program remains silent. If one or more directories could not be found, an error message ("Invalid directory") will be printed on the standard output.

ORIGIN

Hewlett-Packard Company

SEE ALSO

xwcreate(1), xwdestroy(1).

Series 300 and 800 Only

NAME

hpterm - X window system Hewlett-Packard terminal emulator

SYNOPSIS

hpterm [-toolkitoption] [-option]

DESCRIPTION

The *hpterm* program is a terminal emulator for the X Window system. It provides a Term0 compatible terminal for programs that can't use the window system directly.

OPTIONS

The *hpterm* terminal emulator accepts all of the standard X Toolkit command line options along with additional options all of which are listed below (if the option begins with a '+' instead of a '-', the option is restored to its default value):

-b *number*
This option specifies the size of the inner border (the distance between the outer edge of the character and the window border) in pixels. Associated resource: "***borderWidth.**"

-background *color*
This option specifies the color to use for the background of the window. Associated resource: "***background.**"

-bd *color*
This option specifies the color to use for the border of the window. Associated resource: "***borderColor.**"

-bg *color*
This option specifies the color to use for the background of the window. Associated resource: "***background.**"

-borderwidth *number*
This option specifies the width in pixels of the border surrounding the window. Associated resource: "***TopLevelShell.borderWidth.**"

-bw *number*
This option specifies the width in pixels of the border surrounding the window. Associated resource: "***TopLevelShell.borderWidth.**"

-cr *color* This option specifies the color to use for the text cursor. Associated resource: "***cursorColor.**"

-display *display*
This option specifies the X server to contact; see *X*(1). Associated resource: none.

-e *command* [*arguments ...*]
This option specifies the command (and its command line arguments) to be run in the *hpterm* window. The default is to start the user's shell. **This must be the last option on the command line.** Associated resource: none.

-fb *font* This option specifies a font to be used when displaying bold (alternate) text. This font must be the same height and width as the normal (primary) font. If only one of the normal (primary) or bold (alternate) fonts is specified, it will be used for both fonts. Refer to the FONTS section. Associated resource: "***boldFont.**"

-fg *color* This option specifies the color to use for displaying text. Associated resource: "***foreground.**"

-fn *font* This option specifies a font to be used when displaying normal (primary) text. If only one of the normal (primary) or bold (alternate) fonts is specified, it will be used for both fonts. Refer to the FONTS section. Associated resource: "***font.**"

-font *font*
This option specifies a font to be used when displaying normal (primary) text. If only one of the normal (primary) or bold (alternate) fonts is specified, it will be used for both fonts. Associated resource: "***font.**"

-foreground *color*
This option specifies the color to use for displaying text. Associated resource: "***foreground**."

-geometry *geometry*
This option specifies the preferred size and position of the *hpterm* window; see *X*(1). Associated resource: "***term0.geometry**."

-help This option will display a help message. Associated resource: none.

-i This option indicates that *hpterm* should supply the window manager with a bitmapped icon. Associated resource: "**bitmapIcon**."

+i This option indicates that the window manager should generate its own icon for *hpterm*. Associated resource: "**bitmapIcon**."

-iconic This option indicates that *hpterm* should be placed on the display in icon form. Associated resource: "***term0.iconic**."

+iconic This option indicates that *hpterm* should not be placed on the display in icon form. Associated resource: "***term0.iconic**."

-l This option indicates that *hpterm* should send all terminal output to a log file as well as to the screen. Associated resource: "***logging**."

+l This option indicates that *hpterm* should not do logging. Associated resource: "***logging**."

-lf *file* This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (|), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "**HptermLog***XXXXX*" (where *XXXXX* is the process id of *hpterm*) and is created in the directory from which *hpterm* was started (or the user's home directory in the case of a login window). Associated resource: "***logFile**."

-ls This option indicates that the shell that is started in the *hpterm* window should be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile). Associated resource: "***loginShell**."

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be a normal "subshell"). Associated resource: "***loginShell**."

-mb This option indicates that the pointer cursor should be put into blanking mode. In this mode, the cursor will turn on when the pointer is moved, and will be blanked either after a selectable number of seconds or after keyboard input has occurred. The delay is set via the "**pointerBlankDelay**" resource. Associated resource: "***pointerBlank**."

+mb This option indicates that the pointer cursor should remain on. Associated resource: "***pointerBlank**."

-mc *mode*
This option determines how hpterm will generate the foreground color, shadow colors, and shadow tiles of the scrollbar and softkey widgets. Valid modes are "all", "shadow", and "none." Associated resource: "***makeColors**."

-ms *color*
This option specifies the color to be used for the pointer cursor. Associated resource: "***pointerColor**."

-name *name*
This option specifies the application name under which resources are to be obtained, rather than the default executable file name ("hpterm"). Associated resource: "**.name**."

-reverse This option indicates that reverse video should be simulated by swapping the foreground and background colors. Associated resource: "***reverseVideo**."

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors. Associated resource: "***reverseVideo**."

+rv This option indicates that reverse video should not be simulated. Associated resource: "***reverseVideo.**"

-sb This option indicates that a scrollbar should be displayed. Associated resource: "***scrollBar.**"

+sb This option indicates that a scrollbar should not be displayed. Associated resource: "***scrollBar.**"

-sbbg *color*
This option specifies the color to use for the background of the scrollbar window. Associated resource: "***scrollBar.background.**"

-sbfg *color*
This option specifies the color to use for the foreground of the scrollbar window. This value will be ignored if the **makeColors** resource is set to "**all.**" Associated resource: "***scrollBar.foreground.**"

-skbg *color*
This option specifies the color to use for the background of the softkey window. Associated resource: "***softkey.background.**"

-skfg *color*
This option specifies the color to use for displaying softkey text. This value will be ignored if the **makeColors** resource is set to "**all.**" Associated resource: "***softkey.foreground.**"

-skfn *font*
This option specifies a font to be used when displaying softkey text. Associated resource: "***softkey.font.**"

-sl *number*[*suffix*]
This option indicates the number of off screen lines to be saved in the terminal buffer. If no suffix is included or the suffix is "**l**" the total length of the terminal buffer will be *number* plus the length of the terminal window. If the suffix is "**s**" the total length of the terminal buffer will be (*number* plus one) times the length of the terminal window. Associated resource: "***saveLines.**"

-ti *name* This option specifies a name for hpterm to use when identifying itself to application programs. Associated resource: "***termId.**"

-title *name*
This option specifies a window title for hpterm. This string may be used by the window manager when displaying the application. Associated resource: "**.TopLevelShell.title.**"

-tn *name*
This option specifies a name for hpterm to set the "**$TERM**" environment variable to. Associated resource: "***termName.**"

-vb This option indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed. Associated resource: "***visualBell.**"

+vb This option indicates that a visual bell should not be used. Associated resource: "***visualBell.**"

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options. Associated resource: none.

-C This option indicates that the window should receive console output. Associated resource: none.

-L This option indicates that *hpterm* was started by *init*(1m). In this mode, *hpterm* does not try to allocate a new pseudoterminal as *init*(1m) has already done so. In addition, the system program *getty*(1m) is run instead of the user's shell. This option requires a pty name as a separate last argument. **This option should never be used by users when**

starting terminal windows. Associated resource: none.

-S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode, and the file descriptor of the pseudoterminal's master. This allows *hpterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications such as *pam*(1). Associated resource: none.

The following command line arguments are provided for compatibility with older versions. They may not be supported in future releases as the X Toolkit provides standard options that accomplish the same task.

=*geometry*
This option specifies the preferred size and position of the *hpterm* window; see *X*(1). It is equivalent to "**-geometry** *geometry*." Associated resource: "***term0.geometry.**"

#*geometry*
This option specifies the preferred position of the icon window. It is shorthand for specifying the "***iconGeometry**" resource. Associated resource: "**.iconGeometry.**"

-T *string* This option specifies the title for *hpterm*'s window. It is equivalent to "**-title** *string*." Associated resource: "**.TopLevelShell.title.**"

-n *string* This option specifies the icon name for *hpterm*'s windows. It is shorthand for specifying the "***iconName**" resource. Associated resource: "***iconName.**"

-r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to "**-reversevideo**" or "**-rv.**" Associated resource: "***reverseVideo.**"

+r This option indicates that reverse video should no be simulated. It is equivalent to "**+rv.**" Associated resource: "***reverseVideo.**"

-w *number*
This option specifies the width in pixels of the border surrounding the window. It is equivalent to "**-borderwidth** *number*" or "**-bw** *number*." Associated resource: "***TopLevelSHell.borderWidth.**"

RESOURCES

The *hpterm* window consists of an X Toolkit shell widget which contains a term0 widget. The term0 widget contains a scrollbar widget and a softkey widget. Resources specific to the shell widget are:

hpterm Resource Set			
Name	**Class**	**Type**	**Default**
borderColor	BorderColor	Pixel	black
borderWidth	BorderWidth	int	2
geometry	Geometry	string	
iconGeometry	IconGeometry	string	
name	Name	string	hpterm
title	Title	string	Terminal emulator

borderColor
This resource defines the border color of the *hpterm* window.

borderWidth
This resource specifies the width of the *hpterm* window border. This value may be modified by the window manager.

geometry
This resource specifies the preferred size and position of the *hpterm* window.

iconGeometry
This resource specifies the preferred size and position of *hpterm* when iconified. It is not necessarily obeyed by all window managers.

name This resource specifies the name of the instance of the program. It is used when extracting resources from the resource database.

title This resource specifies the window title for *hpterm*. This string may be used by the window manager when displaying this application.

term0 Resource Set			
Name	**Class**	**Type**	**Default**
background	Background	Pixel	white
bitmapIcon	BitmapIcon	Boolean	FALSE
boldFont	Font	string	see FONTS below
copyLine	CopyLine	string	shift right
cursorColor	Foreground	Pixel	black
cut	Cut	string	shift left
flashBorder	FlashBorder	Boolean	FALSE
font	Font	string	see FONTS below
foreground	Foreground	Pixel	black
iconic	Iconic	Boolean	FALSE
internalBorder	BorderWidth	int	2
logFile	LogFile	string	HptermLog*XXXXX*
logging	Logging	Boolean	FALSE
loginShell	LoginShell	Boolean	FALSE
makeColors	MakeColors	string	none
paste	Paste	string	shift middle
pointerBlank	PointerBlank	Boolean	FALSE
pointerBlankDelay	PointerBlankDelay	int	3
pointerColor	Foreground	Pixel	black
reverseVideo	ReverseVideo	Boolean	FALSE
saveLines	SaveLines	string	1s
scrollBar	ScrollBar	Boolean	FALSE
softkeySelect	SoftkeySelect	string	left
termId	TermId	string	X-hpterm
termName	TermName	string	hp2622
visualBell	VisualBell	Boolean	FALSE

background
This resource defines the background color of the text window.

bitmapIcon
This resource defines whether or not *hpterm* will supply the window manager with a bitmapped icon. The supplied bitmap may be ignored by the window manager.

boldFont
This resource defines the font used for bold (alternate) text. See "FONTS" below for defaults.

copyLine
This resource defines the pointer button/modifier combination to be used to activate the CopyLine function. See "POINTER USAGE" below.

cut This resource defines the pointer button/modifier combination to be used to activate the Cut function. See "POINTER USAGE" below.

cursorColor
This resource defines the text cursor color. The pointer cursor color is defined by the **pointerColor** resource.

flashBorder
This resource defines whether or not *hpterm* window border will change color when the pointer cursor enters or leaves the window.

font This resource defines the font used for normal (primary) text. See "FONTS" below for defaults.

foreground
This resource defines the foreground (text) color of the text window.

iconic This resource defines whether or not *hpterm* will start up in iconic form.

internalBorder
This resource defines the number of pixels between the characters and the window border.

logFile This resource defines the name of the file to which a terminal session is logged. The default is "**HptermLog***XXXXX*" (where *XXXXX* is the process id of *hpterm*).

logging This resource defines whether or not a terminal session will be logged.

loginShell
This resource defines whether or not the shell to be run in the window will be started as a login shell (i.e., the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

makeColors
This resource defines how the **bottomShadowColor**, **foreground**, and **topShadowColor** resources of the scrollbar and softkey widgets will be generated. If the value of this resource is "all" then *hpterm* will use the value of the **foreground** resource of the softkey and scrollbar widgets to generate values for the **bottomShadowColor**, **foreground**, and **topShadowColor** resources such that there is a 3-D look. In this case the **topShadowTile** and **bottomShadowTile** are always set to "foreground." If the **makeColors** resource value is "shadow" the **bottomShadowColor** and **topShadowColor** will be generated but **foreground** will not be generated. If the **makeColors** resource value is set to "none" then no colors will be generated.

paste This resource defines the pointer button/modifier combination to be used to activate the Paste function. See "POINTER USAGE" below.

pointerBlank
This resource defines whether or not *hpterm* will put the pointer cursor into blanking mode. In blanking mode, the pointer cursor will turn on when the pointer is moved, and will be blanked either after a selectable number of seconds or after keyboard input has occurred. The delay is set via the **pointerBlankDelay** resource.

pointerBlankDelay
This resource defines the number of seconds to wait before blanking the pointer cursor after the pointer has been moved. When set to "0", the pointer will be blanked only upon keyboard input.

pointerColor
This resource defines the pointer cursor color. The text cursor color is defined by the **cursorColor** resource.

reverseVideo
This resource defines whether or not reverse video will be simulated by swapping the foreground and background colors.

saveLines
This resource defines the number of lines in the terminal buffer beyond the length of the window. The resource value consists of a "*number*" followed by an optional "*suffix*." If no *suffix* is included or the *suffix* is "l" the total length of the terminal buffer will be *number* plus the length of the terminal window. If the suffix is "s" the total length of the terminal buffer will be (*number* plus one) times the length of the terminal window. *Hpterm* **will try to maintain the same buffer to window ratio when the window is resized larger.**

scrollBar
This resource defines whether or not the scrollbar will be displayed.

softkeySelect
This resource defines the pointer button/modifier combination to be used for selecting softkeys. See "POINTER USAGE" below.

termId This resource defines the name for *hpterm* to use when identifying itself to application programs.

termName
This resource defines the string for set the "**$TERM**" environment variable.

visualBell
This resource defines whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received.

The following resources are specified as part of the "softkey" widget (name "softkey", class "Softkey"). For example, the softkey font resource would be specified one of:

```
HPterm*softkey*font:    hp8.8x16
HPterm*Softkey*font:    hp8.8x16
*Softkey*Font:          hp8.8x16
```

Additional resources and information can be found in the **XwPrimitive**(3X) and **CORE**(3X) man pages along with additional information about the various shadow options.

Softkey Resource Set			
Name	**Class**	**Type**	**Default**
background	Background	Pixel	white
bottomShadowColor	Foreground	Pixel	black (see below)
bottomShadowTile	BottomShadowTile	string	foreground (see below)
font	Font	string	(see below)
foreground	Foreground	Pixel	black (see below)
topShadowColor	Background	Pixel	white (see below)
topShadowTile	TopShadowTile	string	50_foreground (see below)

background
This resource defines the background color of the softkey window.

bottomShadowColor
This resource defines the color that is combined with the bottom shadow tile and

foreground color to create a pixmap used to draw the bottom and right sides of the softkey borders. This may be overridden by the term0 **makeColor** resource described above.

bottomShadowTile
This resource defines the tile used in creating the pixmap used for drawing the bottom and right shadows for the softkey borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColor** resource described above.

font This resource defines the font used for softkey text. The softkey font will default to the normal (primary) font of the text window.

foreground
This resource defines the foreground (text) color of the softkey window. This may be overridden by the term0 **makeColor** resource described above.

topShadowColor
This resource defines the color that is combined with the top shadow tile and foreground color to create a pixmap used to draw the top and left sides of the softkey borders. This may be overridden by the term0 **makeColor** resource described above.

topShadowTile
This resource defines the tile used in creating the pixmap used for drawing the top and left shadows for the softkey borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColor** resource described above.

The following resources are specified as part of the "Xwscrollbar" widget (name "scrollBar", class "ScrollBar"). Some example scrollbar resources are:

```
HPterm*scrollBar*initialDelay:    10
HPterm*ScrollBar*RepeatRate:      10
*ScrollBar*Granularity:           1
hpterm*scrollBar*width:           20
```

Additional resources and information can be found in the **XwPrimitive**(3X), **XwScrollBar**(3X), **XwValuator**(3X), and **Core**(3X) man pages along with additional information about the various shadow options.

Scrollbar Resource Set (name "scrollBar", class "ScrollBar")			
Name	**Class**	**Type**	**Default**
background	Background	Pixel	white
bottomShadowColor	Foreground	Pixel	black (see below)
bottomShadowTile	BottomShadowTile	string	foreground (see below)
foreground	Foreground	Pixel	black (see below)
granularity	Granularity	int	2
initialDelay	InitialDelay	int	500
repeatRate	RepeatRate	int	100
topShadowColor	Background	Pixel	white (see below)
topShadowTile	TopShadowTile	string	50_foreground (see below)
width	Width	int	10

background
This resource defines the background color of the scrollbar window.

bottomShadowColor
This resource defines the color that is combined with the bottom shadow tile and foreground color to create a pixmap used to draw the bottom and right sides of the scrollbar borders. This may be overridden by the term0 **makeColor** resource described above.

bottomShadowTile
This resource defines the tile used in creating the pixmap used for drawing the bottom and right shadows for the scrollbar borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColor** resource described above.

foreground
This resource defines the foreground color of the scrollbar window. This may be overridden by the term0 **makeColor** resource described above.

granularity
This resource defines the number of lines to advance the slider when the button is being held down on an arrow. The value is defined in milliseconds.

initialDelay
This resource defines the delay to wait between the time the button is held down on an arrow before the slider starts its repetitive movement. The value is defined in milliseconds.

repeatRate
This resource defines the continuous repeat rate to use to move the slider while the button is being held down on an arrow. The value is also defined in milliseconds.

topShadowColor
This resource defines the color that is combined with the top shadow tile and foreground color to create a pixmap used to draw the top and left sides of the scrollbar borders. This may be overridden by the term0 **makeColor** resource described above.

topShadowTile
This resource defines the tile used in creating the pixmap used for drawing the top and left shadows for the scrollbar borders. Valid tile names are described in **XwCreateTile**(3X). This may be overridden by the term0 **makeColor** resource described above.

width This resource defines the width of the scrollbar in pixels.

POINTER USAGE

Hpterm allows you to cut and paste text within its own or other windows. All cutting and pasting is done to/from the first global cut buffer.

The default button assignments may be changed via various resource strings. The default button functions are all activated when the "shift" key is pressed. The cut and paste functions and their default button assignments are:

Enter The left hand button "cuts" the text from the pointer (at button release) through the end of line (including the new line), saving it in the cut buffer, and immediately "pastes" the line, inserting it as keyboard input. This provides a history mechanism.

Cut The center button is used to "cut" text into the cut buffer. Move the pointer to the beginning of the text to cut, press the button, move the cursor to the end of the region, and release the button. The "cut" text will not include the character currently under the pointer.

Paste The right hand button "pastes" the text from the cut buffer, inserting it as keyboard input.

The enter, cut, and paste key functions can be configured to any button and modifier combination desired via various resources. Each assignment consists of an optional combination of modifiers ("none" or any combination of "shift", "meta", "lock", "control", "mod1", ..., "mod5" separated by blanks), followed by a "|" and the name of the button ("left", "middle", "right", "button1", ..., "button5"). For example, if it is desired for the cut function to be associated with the middle

button with shift and control pressed, one could use the following resource line:

*cut: shift control | middle

For a full list of resource names, see "RESOURCES" above.

FONTS

Hpterm will try to select default fonts which match your display and your keyboard language. If the normal (primary) and bold (alternate) fonts are specified, they will be used. If only one is specified (via either command line or resources), it will be used for both the normal (primary) and bold (alternate) fonts. If neither normal (primary) or bold (alternate) fonts are specified, *hpterm* trys to find them based on the "**$LANG**" environment variable. It looks in "*/usr/lib/nls/$LANG/X11font*" directory for fonts linked to fonts in the normal X11 font directory "*/usr/lib/X11/fonts*." These fonts are "*base.low*" and "*alt.low*" for displays whose width is less than or equal to 640 pixels or whose height is less than or equal to 400 pixels, "*base.med*" and "*alt.med*" for displays whose width is greater than 640 and less than or equal to 1024 pixels or whose height is greater than 400 and less than or equal to 800 pixels, and "*base.high*" and "*alt.high*" for displays whose width is greater than 1024 pixels and whose height is greater than 800 pixels. If "**$LANG**" is not defined, it will use the language corresponding to the language of your keyboard. If these font files can not be found, *hpterm* will then try to find them in the "*/usr/lib/nls/n-computer/X11font*" directory. If these fonts can not be found, the fonts "fixed" and "bold" in the font path specified in xset -fp will be used. (Note: Font files reside on the machine on which the server is running.)

Control-N will switch to the bold (alternate) font and control-O will switch back to the normal (primary) font. *Hpterm* will switch back to the normal (primary) font automatically at the beginning of each line.

Hpterm will try to use the appropriate keymap to match the type of keyboard being used. It currently supports 21 different language versions of the HP keyboard.

ENVIRONMENT

Hpterm sets the environment variable "**$TERM**" properly for the size window you have created. It sets "**$LINES**" and "**$COLUMNS**" to be the number of lines and columns of the terminal screen. It also uses and sets the environment variable "**$DISPLAY**" to specify its server connection. The *resize*(1) command may be used to reset "**$LINES**" and "**$COLUMNS**" after the window size has been changed.

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1), resize(1), xset(1), xterm(1), pty(4), Core(3X), XwScrollBar(3X), XwPrimitive(3X), XwCreateTile(3X), XwValuator(3X), XwArrow(3X)

NAME

hpwm - the Hewlett Packard window manager for X

SYNOPSIS

hpwm [options]

DESCRIPTION

The Hewlett-Packard Window Manager (*hpwm*) is an X11 client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window state such as placement, size, icon/normal display, input focus ownership etc. It also provides session management functions such as stopping a client.

When *hpwm* is invoked, it retrieves configuration resource values from the following files.

/usr/lib/X11/app-defaults/Hpwm
.Xdefaults
hpwm resource description file (.hpwmrc)

OPTIONS

-display *display*
This option specifies the display to use; see *X(1)*.

-xrm *resourcestring*
This option specifies a resource string to use.

X DEFAULTS

Hpwm is configured from its resource database. This database is built from the resource specifications in the *.Xdefaults* and */usr/lib/X11/app-defaults/Hpwm* resource files. Entries in these files may refer to other resource files that specify specific types of resources (e.g., bitmaps and menus). If the same resource is specified in more than one of the resource files the resource specification in the *.Xdefaults* file has precedence over the specification in the */usr/lib/X11/app-defaults/Hpwm* file. Hpwm has built-in default values for all the resources that it uses (refer to the descriptions of specific resources).

Hpwm is the resource class name of hpwm and **hpwm** is the resource name used by hpwm to look up resources. In the following discussion of resource specification "Hpwm" and "hpwm" can be used interchangeably.

Hpwm uses the following types of resources:

General Appearance Resources:

These resources are used to specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, client window frames and icons.

Specific Appearance And Behavior Resources:

These resources are used to specify hpwm appearance and behavior (e.g., window management policies). They are not set separately for different hpwm user interface components.

Client Class Specific Resources:

These hpwm resources can be set for a particular class of client windows. They specify class-specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (e.g., "foreground") or a resource class (e.g., "Foreground"). If the value of a resource is a filename and if the filename is prefixed by "~/" then it is relative to the path contained in the *$HOME* environment variable (generally the user's home directory).

General Appearance Resources

The syntax for specifying *general appearance resources* that apply to window manager icons, menus

and client window frames is:

"Hpwm*<resource_id>"

For example, "Hpwm*foreground" is used to specify the foreground color for hpwm menus, icons, client window frames.

The syntax for specifying *general appearance resources* that apply to a particular hpwm component is:

"Hpwm*[menu|icon|client]*<resource_id>"

If *menu* is specified the resource is applied only to hpwm menus, if *icon* is specified the resource is applied to icons, and if *client* is specified the resource is applied to client window frames. For example, "Hpwm*icon*foreground" is used to specify the foreground color for hpwm icons, "Hpwm*menu*foreground" specifies the foreground color for hpwm menus, and "Hpwm*client*foreground" is used to specify the foreground color for hpwm client window frames.

The following *general appearance resources* can be specified:

activeBackground (class **Background**)
Specifies the background color of the hpwm decoration when the window is active (has the keyboard focus). This resource can have any legal color as a value. The default value is the *background* general appearance resource value.

activeBackgroundTile (class **ActiveBackgroundTile**)
This resource specifies the background tile of the hpwm decoration when the window is active (has the keyboard focus). This resource can be any legal HP X Widget tile value (See *XwCreateTile*(*3X*)). The default value is "background".

activeBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *bottomShadowColor* general appearance resource value.

activeBottomShadowTile (class **BottomShadowTile**)
This resource specifies the bottom shadow tile of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *bottomShadowTile* general appearance resource value.

activeForeground (class **Foreground**)
This resource specifies the foreground color of the hpwm decoration when the window is active (has the keyboard focus). This resource can have any legal color as a value. The default value is the *foreground* general appearance resource value.

activeTopShadowColor (class **Background**)
This resource specifies the top shadow color of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *topShadowColor* general appearance resource value.

activeTopShadowTile (class **TopShadowTile**)
This resource specifies the top shadow tile of the hpwm decoration when the window is active (has the keyboard focus). The default value is the *topShadowTile* general appearance resource value.

background (class **Background**)
This resource specifies the background color. Any legal X color may be specified. The default is "White".

backgroundTile (class **BackgroundTile**)
This resource specifies the background tile of the hpwm decoration when the window is inactive (does not have the keyboard focus). This resource can be any legal HP X Widget tile value. The default value is "25_foreground".

bottomShadowColor (class **Foreground**)
This resource specifies the top shadow color. This color is used for the lower and right

bevels of the window manager decoration. Any legal X color may be specified. The default is "Black".

bottomShadowTile (class **BottomShadowTile**)
This resource specifies the bottom shadow tile. This tile is used for the lower and right bevels of the window manager decoration. Any legal HP X Widget tile may be specified. The default is "foreground".

font (class **Font**)
This resource specifies the font used in menus, window titles, and icon labels. Any available X font may be specified. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class **Foreground**)
This resource specifies the foreground color. Any legal X color may be specified. The default is "Black".

makeActiveColors (class **MakeColors**)
If the value of this resource is "all" (or "true") then hpwm will use the value of the **activeBackground** resource to make values for the **activeBottomShadowColor, activeForeground** and **activeTopShadowColor** resources that provide a 3-D appearance. In this case the **activeTopShadowTile** and **activeBottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" (or "false") then no colors will be automatically made. The default value for this resource is "shadow".

makeColors (class **MakeColors**)
If the value of this resource is "all" then hpwm will use the value of the **background** resource to make values for the **bottomShadowColor, foreground** and **topShadowColor** resources that provide a 3-D appearance. In this case the **topShadowTile** and **bottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" then no colors will be automatically made. The default value for this resource is "shadow".

topShadowColor (class **Background**)
This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. Any legal X color may be specified. The default is "White".

topShadowTile (class **TopShadowTile**)
This resource specifies the top shadow tile. This tile is used for the upper and left bevels of the window manager decoration. Any legal HP X Widget tile may be specified. The default is "50_foreground".

Specific Appearance And Behavior Resources

The syntax for specifying *specific appearance and behavior resources* is:

"**Hpwm***<resource_id>"

For example, "Hpwm*keyboardFocusPolicy" is used to specify the window manager policy for setting the keyboard focus to a particular client window.

The following *specific appearance and behavior resources* can be specified:

bitmapDirectory (class **BitmapDirectory**)
This resource identifies a directory that is to be searched for bitmaps that are referenced by hpwm resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is "/usr/include/X11/bitmaps".

buttonBindings (class **ButtonBindings**)
This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the *hpwm resource description file* file.

The default value for this resource is "DefaultButtonBindings".

colormapFocusPolicy (class **ColormapFocusPolicy**)
This resource indicates the colormap focus policy that is to be used. If the resource value is "explicit" then a colormap selection action is done on a client window to set the colormap focus to that window. If the value is "pointer" then the client window that contains the pointer will have the colormap focus. If the value is "keyboard" then the client window that has the keyboard input focus will have the colormap focus. The default value for this resource is "keyboard".

configFile (class **ConfigFile**)
The resource value is the pathname for an *hpwm resource description file*. The default is **.hpwmrc** in the user's home directory, if this file exists, otherwise **/usr/lib/X11/system.hpwmrc**.

doubleClickTime (class **DoubleClickTime**)
This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is "500" (ms).

iconAutoPlace (class **IconAutoPlace**)
This resource indicates whether icons are automatically placed on the screen by hpwm. If the resource value is "True" then hpwm does automatic icon placement. Users may specify an initial icon position and can move icons, but hpwm will adjust the user-specified position to fit the icon placement scheme (refer to the **IconPlacement** resource). If the resource value is "False" then hpwm does not do automatic icon placement, and the icon placement scheme is ignored. The default value of this resource is "True".

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is "label" (only the label part is displayed) or "image" (only the image part is displayed) or "label image" (both the label and image parts are displayed). A value of "activelabel" can also be specified as an enhancement to the "label" value to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration is that each icon has a label part and an image part ("label image").

iconImageMaximum (class **IconImageMaximum**)
This resources specifies the maximum size of the icon *image*. The resource value is <width>x<height> (e.g., "64x64"). The default value of this resource is "50x50".

iconImageMinimum (class **IconImageMinimum**)
This resources specifies the minimum size of the icon *image*. The resource value is <width>x<height> (e.g., "32x50"). The default value of this resource is "32x32".

iconPlacement (class **IconPlacement**)
This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

<primary_layout> <secondary_layout>

The layout values are one of the following:

top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (vertical) layout value should not be used for both the primary_layout and the secondary_layout (e.g., don't use "top" for the primary_layout and "bottom" for the secondary layout). The primary_layout indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The secondary_layout indicates where to place new rows or columns. For example, "top right" indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement (compatible

with the PM placement policy) is "left bottom" (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen).

iconPlacementMargin (class **IconPlacementMargin**)
If nonnegative, this resource specifies the distance between the edge of the screen and the icons that are placed along the edge of the screen. Otherwise, this distance is set equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column). The default value for this resource is -1.

interactivePlacement (class **InteractivePlacement**)
This resource controls the initial placement of new windows on the screen. If it is "True", then the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper-left hand corner of the window. If "False" then windows will be placed according to the initial window configuration attributes. The default value of this resource is "False."

keyBindings (class **KeyBindings**)
This resource identifies the set of key bindings for window management functions. The named set of key bindings is specified in *hpwm resource description file* file. The default value for this resource is "DefaultKeyBindings".

keyboardFocusPolicy (class **KeyboardFocusPolicy**)
If set to "pointer" the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that hpwm adds). If set to "explicit" the policy is to have the keyboard focus set to a client window when the user does a select (Button1 Down) on the client window or any part of the associated hpwm decoration. The default value for this resource is "explicit".

limitResize (class **LimitResize**)
If this resource is "True" the user is not allowed to resize a window to greater than the maximum size (set by the **maximumClientSize** resource or using the WM_NORMAL_HINTS window property). The default value for this resource is "False".

maximumMaximumSize (class **MaximumMaximumSize**)
This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is <width>x<height> (e.g., "1024x1024") where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveThreshold (class **moveThreshold**)
This resource is used to control the sensitivity of "dragging" operations that are used in moving windows and icons. The value of this resource is the number of pixels that the locator will be moved with a button down before the move operation will be initiated. This is used to prevent window/icon movement when a click or double-click is done and there is unintentional pointer movement with the button down. The default value of this resource is "4" (pixels).

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not the keyboard input focus selection button press (if keyboardFocusPolicy is "explicit") is passed on to the client window or used to do a window management action associated with the window decorations. If the resource value is "False" then the button press will not be used for any operation other than selecting the window to be the keyboard input focus; if the value is "True" the button press will be passed to the client window or used to do a window management operation if appropriate. The default value for this resource is "True".

positionIsFrame (class **PositionIsFrame**)
This resource indicates how client window position information (from the WM_NORMAL_HINTS property and from configuration requests) is to be interpreted.

If the resource value is "True" then the information is interpreted as the position of the hpwm client window frame, if the value is "False" then it is interpreted as being the position of the window. The default value of this resource is "True".

positionOnScreen (class **PositionOnScreen**)
This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is "True"). If a window is larger then the size of the screen then at least the upper left corner of the window will be on-screen. If the resource value is "False" then windows will be placed in the requested position even if totally off-screen. The default value of this resource is "True".

quitTimeout (class **QuitTimeout**)
This resource specifies the amount of time (in milliseconds) that hpwm will wait for a client to update the WM_COMMAND property after hpwm has sent the WM_SAVE_YOURSELF message. This protocol will only be used for those clients that have a WM_SAVE_YOURSELF atom in the WM_PROTOCOLS client window property. The default value of this resource is "1000" (ms).

resizeBorderWidth (class **ResizeBorderWidth**)
This resource specifies the width (in pixels) of the border around client windows. There will always be some visible border even if this value is set to 0. The default is "10" (pixels).

resizeCursors (class **ResizeCursors**)
This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If "True" the cursors are shown, otherwise the window manager cursor is shown. The default value is "True".

transientDecoration (class **TransientDecoration**)
This controls the amount of decoration that Hpwm puts on transient windows. The gadget specification is exactly the same as for the **clientDecoration** resource (see client class specific resources section below). Transient windows are identified by the WM_TRANSIENT_FOR property which is added by the client to indicate a relatively temporary window. By default, Hpwm will decorate transient windows with a title bar and no other gadgets. The default value for this resource is "title."

Client Class Specific Resources

The syntax for specifying *client class specific resources* for specific classes of clients is:

"**Hpwm***<client_class>.<resource_id>"

For example, "Hpwm*HPterm.systemMenu" is used to specify the system menu to be used with hpterm clients.

The syntax for specifying *client class specific resources* for all classes of clients is:

"**Hpwm***<resource_id>"

Specific client class specifications take precedence over the specifications for all client classes. For example, "Hpwm*systemMenu" is used to specify the system menu to be used for all classes of clients that don't have a specific system menu specified.

The syntax for specifying resource values for windows that have an unknown class (i.e. the window does not have a WM_CLASS property associated with it) is:

"**Hpwm*defaults***<resource_id>"

For example, "Hpwm*defaults*iconImage" is used to specify the icon image to be used for windows that have an unknown class. This is also how a default icon image can be specified for windows that do not have an icon image available from any other source.

The following *client class specific resources* can be specified:

clientDecoration (class **ClientDecoration**)

This resource controls the amount of gadgetry in the client window frame. The resource is specified as a list of gadgets to specify their inclusion in the frame. If a gadget is preceded by a minus sign, then that gadget is excluded from the frame. The *sign* of the first item in the list determines the initial amount of gadgetry. If the sign of the first item is minus, then hpwm assumes all gadgets present and starts subtracting from that set. If the sign of the first item is plus (or not specified), then hpwm starts with no gadgets and builds up a list from the resource.

Name	Description
all	Include all gadgets
maximize	Maximize box (includes title bar)
minimize	Minimize box (includes title bar)
none	No gadgets
resize	Resize border
system	System menu box (includes title bar)
title	Title bar (only)

The default value for this resource is "all."

iconImage (class **IconImage**)

This resource can be used to specify an icon image for a particular class of clients (i.e. "Hpwm*<client_class>.iconImage") or an icon image to be used for all classes of clients that don't have a specifically specified icon image (i.e. "Hpwm*iconImage"). The resource value is a pathname for a bitmap file. If specified this resource overrides any client specified icon image.

The default value is to display the client supplied icon image if it is defined; if a client icon image is not available then the icon image specified by "Hpwm*defaults*iconImage" is used if it is specified; if an icon image is not supplied by the user or the client then a built-in window manager icon image is used.

iconImageBackground (class **Background**)

This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (i.e. specified by "Hpwm*background or Hpwm*icon*background).

iconImageBottomShadowColor (class **Foreground**)

This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (i.e. specified by Hpwm*bottomShadowColor or Hpwm*icon*bottomShadowColor).

iconImageBottomShadowTile (class **BottomShadowTile**)

This resource specifies the bottom shadow tile of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow tile (i.e. specified by Hpwm*bottomShadowTile or Hpwm*icon*bottomShadowTile).

iconImageForeground (class **Foreground**)

This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon foreground color (i.e. specified by "Hpwm*foreground or Hpwm*icon*foreground).

iconImageTopShadowColor (class **Background**)

This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (i.e. specified by Hpwm*topShadowColor or Hpwm*icon*topShadowColor).

iconImageTopShadowTile (class **TopShadowTile**)

This resource specifies the top shadow tile of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow tile (i.e.

specified by Hpwm*topShadowTile or Hpwm*icon*topShadowTile).

makeIconColors (class **MakeColors**)
If the value of this resource is "all" then hpwm will use the value of the **iconImageBackground** resource to make values for the **iconImageBottomShadowColor**, **iconImageForeground** and **iconImageTopShadowColor** resources that provide a 3-D appearance. In this case the **iconImageTopShadowTile** and **iconImageBottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" then no colors will be automatically made. The default value for this resource is "shadow".

makeMatteColors (class **MakeColors**)
If the value of this resource is "all" then hpwm will use the value of the **matteBackground** resource to make values for the **matteBottomShadowColor, matteForeground** and **matteTopShadowColor** resources that provide a 3-D appearance. In this case the **matteTopShadowTile** and **matteBottomShadowTile** are always set to "foreground". If the makeColors resource value is "shadow" the top and bottom shadow colors will be made but the foreground color will not be made. If the makeColors resource value is "none" then no colors will be automatically made. The default value for this resource is "shadow".

matteBackground (class **Background**)
This resource specifies the background color of the matte, when **matteWidth** is positive. The default value of this resource is the client background color (i.e. specified by "Hpwm*background or Hpwm*client.background).

matteBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow color (i.e. specified by "Hpwm*bottomShadowColor or Hpwm*client.bottomShadowColor).

matteBottomShadowTile (class **BottomShadowTile**)
This resource specifies the bottom shadow tile of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow tile (i.e. specified by "Hpwm*bottomShadowTile or Hpwm*client.bottomShadowTile).

matteForeground (class **Foreground**)
This resource specifies the foreground color of the matte, when **matteWidth** is positive. The default value of this resource is the client foreground color (i.e. specified by "Hpwm*foreground or Hpwm*client.foreground).

matteTopShadowColor (class **Background**)
This resource specifies the top shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow color (i.e. specified by "Hpwm*topShadowColor or Hpwm*client.topShadowColor).

matteTopShadowTile (class **TopShadowTile**)
This resource specifies the top shadow tile of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow tile (i.e. specified by "Hpwm*topShadowTile or Hpwm*client.topShadowTile).

matteWidth (class **MatteWidth**)
This resource specifies the width of the optional *matte* that can be specified to frame the client area of the window. The matte sits just inside the other client window frame decoration. This is useful to help old applications fit into a suite of new applications that use the HP widget set. The matte is given a 3-D effect just like the HP widgets. The default value is 0, which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This is a size specification that indicates the client size to be used when an application is maximized. The resource value is specified as "<width>x<height>". If this resource is not specified then the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size where the client window with window

management borders fills the screen.

systemMenu (class **SystemMenu**)
This resource indicates the name of the menu pane that is posted when the system menu is popped up (usually by pressing button 1 on the system box gadget on the client window frame). Menu panes are specified in the *hpwm resource description file* file. System Menus can be customized on a client class basis by specifying resources of the form Hpwm*<client class>.systemMenu (See **Hpwm Resource Description File Syntax**). The default value of this resource is "DefaultSystemMenu".

RESOURCE DESCRIPTION FILE

The *hpwm resource description file* is a supplementary resource file that contains resource descriptions that are referred to by entries in the defaults files (.Xdefaults, app-defaults/Hpwm). It contains descriptions of resources that are to be used by hpwm, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular *hpwm resource description file* can be selected using the **configFile** resource.

The following types of resources can be described in the *hpwm resource description file*:

Buttons	Window manager functions can be bound (associated) with button press events.
Keys	Window manager functions can be bound (associated) with key press events.
Menu	These menu panes can be used for the system menu and other menus posted with key and button bindings.

Hpwm Resource Description File Syntax

The *hpwm resource description file* is a standard text file that contains items of information separated by blanks, tabs and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (e.g., the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them by the back-slash character (\). All text from an unquoted # to the end of the line is regarded as a comment and is not interpreted as part of a resource description. Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for botton and key binding sets, and menu panes. The function specification has the following syntax:

```
function =          function_name [function_args]
function_name =     <window manager function>
function_args =     {quoted | unquoted_item}
```

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by hpwm as *f.nop*.

f.beep	This function beeps.
f.circle_down	This function lowers the highest mapped client window that partially or completely obscures another client window to the bottom of the window stack (where it obscures no other window).
f.circle_up	This function raises the lowest mapped client window that is partially or completely obscured by another client window to the top of the window stack (where it is obscured by no other window).
f.exec or !	This function causes *function-args* to be executed (using */bin/sh*).
f.focus_color	This function sets the colormap focus to a client window. If this function is done in a root context then the default colormap (setup by the *X Window System* for the screen where hpwm is running) will be installed and there will be no specific client window colormap focus. This function is treated as *f.nop* if *colormapFocusPolicy* is not "explicit".

f.focus_key This function sets the keyboard input focus to a client window or icon. This function is treated as *f.nop* if *keyboardFocusPolicy* is not "explicit" or the function is executed in a root context.

f.kill This function causes a client's X connection to be terminated (usually resulting in termination of the client).

f.lower This function lowers a client window to the bottom of the window stack (where it obscures no other window).

f.maximize This function causes a client window to be displayed with its maximum size.

f.menu If this function appears in a menu pane entry, it associates the cascading (pull-right) menu identified by function_args with the menu pane entry. If this function appears in a key or button binding, it posts the menu identified by function_args.

f.minimize This function causes a client window to be minimized / iconized.

f.move This function allows a client window to be interactively moved.

f.next_cmap This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen).

f.nop This function does nothing.

f.normalize This function causes a client window to be displayed with its normal size.

f.post_smenu This function is used to post the system menu.

f.prev_key This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen).

f.quit_hpwm This function terminates hpwm (but NOT the X window system).

f.raise This function raises a client window to the top of the window stack (where it is obscured by no other window).

f.raise_lower This function raises an obscured client window to the top of the window stack (where it is obscured by no other window) and lowers a client window that is on top of the window stack to the bottom of the stack (where it obscures no other window).

f.refresh This function causes all windows to be redrawn.

f.refresh_win This function causes a client window to be redrawn. It is the same as *f.refresh* if it is used in a menu posted with a root context.

f.resize This function allows a client window to be interactively resized.

f.restart This function causes hpwm to be restarted (effectively terminated and re-exec'ed).

f.separator This function causes a menu separator to be put in the menu pane entry (the label is ignored).

f.title This function inserts a title in the menu pane if it is the first entry in the menu pane description. By default a menu pane has no title.

Each function may be constrained as to which resource types can specify the function (e.g., menu pane), and also what context the function can be used in (e.g., the function is done to the selected client window). Function contexts are:

root No client window or icon has been selected as an object for the function.

window A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the

window is in its normalized state (e.g., f.maximize) or its maximized state (e.g., f.normalize).

icon An icon has been selected as an object for the function.

frame A client window frame has been selected as an object for the function. This includes the window frame's title bar.

title A client window title bar has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as *f.nop*. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	Contexts	Resources
f.beep	root,icon,window	button,key,menu
f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	icon,window	button,key,menu
f.kill	icon,window	menu
f.lower	icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.post_smenu	root,icon,window	button,key
f.prev_key	root,icon,window	button,key,menu
f.quit_hpwm	root	menu
f.raise	icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu
f.resize	window	button,key,menu
f.restart	root	menu
f.separator	root,icon,window	menu
f.title	root,icon,window	menu

Button Bindings

A window manager function can be done with a pointer button press or release when the pointer is over a client window, an icon or the root window. The context for indicating where the button action applies is also the context for invoking the window manager function when the button press is done.

The button binding syntax is:

```
Buttons bindings_set_name
{
  button   context   function
  button   context   function
                .
                .
  button   context   function
```

}

The *button* specification is done using the syntax supported by the *X Toolkit's* translation manager, with three exceptions: only a single event may be specified, the event must be a *ButtonPress* or a *ButtonRelease*, and all modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). For example, button 1 down with the [Shift] key pressed is specified by: Shift<Btn1Down> and button 1 up with the [Extend char] key pressed is specified by Meta<Btn1Up>. Button release specifications are interpreted by hpwm as a "click" (i.e. a button press followed by a button release with less than the move threshold amount of motion in between).

The syntax for the *context* specification is:

```
context =   object["|"context]
object =    root | icon | window | title | frame
```

The context specification indicates where the pointer must be for the button binding to be effective. The *frame* context is for the client window frame and the *title* context is for the title area of the client window frame. For example, a context of *window* indicates that the pointer must be over a client window or window frame or window title for the button binding to be effective. For button bindings the *frame* and *title* contexts are equivalent to the *window* context.

If a *f.nop* function is specified for a button binding the button binding will not be done.

Key Bindings

A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings.

The key binding syntax is:

```
Keys bindings_set_name
{
  key   context   function
  key   context   function
          .
          .
  key   context   function
}
```

The *key* specification is done using the syntax supported by the *X Toolkit's* translation manager, with three exceptions: only a single event may be specified, the event must be a key event, and all modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). For example, a shifted [C] key press is specified by Shift<Key>c.

If a *f.nop* function is specified for a key binding the key binding will not be done. If a *f.post_smenu* or *f.menu* function is bound to a key, hpwm will automatically use the same key for removing the menu from the screen after it has been popped up.

The *context* specification syntax is the same as for button bindings. For key bindings the *frame* and *title* contexts are equivalent to the *window* context.

Menu Panes

The context for window manager functions that are done from the system menu is *icon* or *window* depending on where the system menu was popped up. For other menus the context depends on the location of the pointer (for menus posted by button bindings) or the location of the keyboard input focus (for menus posted by key bindings).

The menu pane specification syntax is:

```
Menu menu_pane_name
{
  label   function
  label   function
          .
          .
  label   function
}
```

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. The *label* may be a string or a bitmap file. The label specification has the following syntax:

```
label =          text | bitmap_file
bitmap_file =    "@"<file_name>
text =           quoted_item | unquoted_item
```

The sting encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the *f.nop* function or an invalid function or a function that doesn't apply in the current context.

WINDOW MANAGER COMMANDS

The user interactively commands the window manager with the keyboard and pointer. This interface can be configured by the user by changing entries in the resource files for hpwm (see above).

Pointer Commands

The pointer (usually a mouse) is the device that controls a cursor that ranges over the entire screen. Hpwm uses the pointer in the following ways:

Button 1 Click
Select an object/action. The default actions performed by doing a button 1 click on a window manager object are:

Object	Action
window frame	top window
icon	top icon
window frame and window	explicit keyboard focus selection
icon	explicit keyboard focus selection
frame minimize gadget	minimize the window
frame maximize gadget	maximize the window

Button 1 Double-click
The default action performed by doing a button 1 double-click on an icon is to normalize the associated window.

Button 1 Drag
Select and perform a move/resize action or pop up a menu and select a menu item. The default actions performed by doing a button 1 drag on a window manager object are:

Object	Action
window frame title	move the window

window resize border	resize the window
icon	move the icon
frame system gadget	popup the system menu, select an item

The **buttonBindings** resource can be used to associate window management functions with button presses (although default, unmodified Button 1 function bindings supercede unmodified Button 1 *buttonBindings* specifications). This is useful for users who do not want screen space taken up by window decoration. This mechanism allows them to run without decoration, but still have a way of manipulating windows. The default button bindings are:

Button	Function	Context	Description
Button1	f.menu	root	post the default root window menu
Button1 Click	f.raise	frame	raise window to top of stack
Meta-Button1 Drag	f.move	window	move a window
Meta-Button3 Click	f.minimize	window	minimize a window

Keyboard Commands

Keyboard bindings for window management functions can be defined to allow window management to be done without using a pointer. The resource **keyBindings** is used to associate key presses with window management functions. The default key bindings for window manipulation are:

Key	Function	Description
Shift-Escape	f.post_smenu	pop up the system menu
Meta-Tab	f.next_key	go to next window in stack
Return	<built-in>	accept menu selection
Up-Arrow	<built-in>	move to previous item in menu
Down-Arrow	<built-in>	move to next item in menu

Key bindings that are used to pop up the system menu are automatically used by hpwm as key bindings for removing the menu when it is popped up.

The f.next_key function is only valid when **keyboardFocusPolicy** is **explicit**. No such function is provided when **keyboardFocusPolicy** is **pointer**.

Interactive Window Placement

The user has the option of interactively positioning and sizing new windows before they appear on the display. When **interactivePlacement** resource is set to "True", the pointer shape will change before the new window is mapped. A window will appear in the center of the screen that provides position and size feedback.

The user sets the initial position by moving the pointer to the desired location and clicking the Select button. The user may set the initial size of the window by depressing the Select button to fix the location of the upper left-hand corner of the window and then dragging the pointer to the desired position of the lower right-hand corner of the window. The window size will be fixed once the user releases the Select button. The **moveThreshold** resource will be used to distinguish between an accidental movement of the pointer while clicking versus a deliberate size setting action.

You can also do interactive placement strictly from the keyboard. The arrow keys move the pointer in the expected direction. Arrow keys while the [CTRL] key is held down to moves the pointer in large increments. The space bar stops moving the window and gets hpwm ready to change its size. The [Return] key completes interactive placement at any time.

The feedback window provides position and size information as the pointer is moved. The size feedback will be in multiples of width_inc and height_inc if those values are defined in the WM_NORMAL_HINTS for the window. In addition, a wire frame of the window is drawn to give

the user graphic feedback of the window size.

If the window position information is specified when the client is invoked, then interactive placement will not take place. The window will be placed at the position specified. The window will be of the default size unless the user also specifies this along with the position. If the user only specifies the size of the window, hpwm will use that size as the default during interactive placement.

Hpwm will not do interactive placement on windows that have the WM_TRANSIENT_FOR property set. These are assumed to exist for short duration interactions (dialog boxes) with which interactive placement would interfere.

ENVIRONMENT

Hpwm uses the environment variable **$HOME** specifying the user's home directory.

FILES

/usr/lib/X11/system.hpwmrc
/usr/lib/X11/app-defaults/Hpwm
$HOME/.Xdefaults

COPYRIGHT

Copyright 1988, Hewlett Packard Company

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

resize - reset shell parameters to reflect the current size of a window

SYNOPSIS

resize [-option ...]

DESCRIPTION

Resize prints on its standard output the commands for setting $TERM, $LINES, and $COLUMNS for a shell to reflect the current size of its window. The $SHELL environment variable is used to determine the shell for which to form the commands. The $TERM environment variable is used to determine the escape sequences to be used to determine the window size. Both of these can be overridden by command line options.

Resize is never executed directly, but should be run via *eval*(1) similar to *tset*(1) to cause the shell to execute the commands. For example, the following functions will reset the environment of the current shell for *sh*(1) and *ksh*(1):

```
xs()    { eval 'resize'; }
xrs()   { eval 'resize -s $@'; }
```

An equivalent for *csh*(1) is:

```
alias xs  'set noglob; eval `resize`'
alias xrs 'set noglob; eval `resize -s \!\*`'
```

OPTIONS

The *resize* program accepts the following options listed below:

-c This option indicates that *resize* should format its commands for *csh*(1).

-h This option indicates that *resize* should use Hewlett Packard terminal escape sequences to obtain the terminal's new window size.

-s [row col]
This option indicates that *resize* should use Sun escape sequences to obtain the terminal's new window size. In this mode of operation, a new row and column size may be specified on the command line.

-u This option indicates that *resize* should format its commands for *sh*(1) or *ksh*(1).

-x This option indicates that *resize* should use VT102 escape sequences to obtain the terminal's new window size.

FILES

$HOME/.profile	*sh*(1) and *ksh*(1) user's functions for *resize*.
$HOME/.cshrc	*csh*(1) user's alias for *resize*.

NOTES

"-s" must be the last option on the command line when specified.

There should be some global notion of display size; termcap and terminfo need to be rethought in the context of window systems.

ORIGINS

MIT Distribution

SEE ALSO

sh(1), ksh(1), csh(1), eval(1), hpterm(1), tset(1), xterm(1)

Series 300 and 800 Only

NAME

rgb - X Window System color database creator.

SYNOPSIS

rgb [*filename*]

DESCRIPTION

rgb creates a data base used by the X window system server for its colors. Stdin is used as its input and must be in the format of:

0-255 0-255 0-255 colorname

For example:

```
  0   0   0   black
  0 128   0   green
255 255 255   white
```

rgb stands for *r*ed-*g*reen-*b*lue. Each element can have no intensity (0) to full intensity (255). How the elements are combined determines the actual color. The name given to the color can be descriptive or fanciful.

In other words, the sequence:

0 0 128

can be given the name 'blue' or 'unicorn blue'. There can also be two (or more) entries with the same element numbers or names.

ARGUMENTS

filename If filename is given, *rgb* produces two files; *filename.dir* and *filename.pag*. Otherwise the default filename is */usr/lib/X11/rgb*.

ORIGIN

MIT Distribution

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

sb2xwd - translate Starbase bitmap to xwd bitmap format

SYNOPSIS

sb2xwd

DESCRIPTION

This command translates a bitmap file created by one of the Starbase bitmap-to-file procedures into an *XWD* format file. The *XWD* format is defined by the *xwd(1)* and *xwud(1)* X window dump utility programs. The Starbase bitmap file format is described in *bitmapfile(4)*. Translation is done from standard input to standard output.

Starbase bitmaps created in *pixel-major* format will be translated into *ZPixmap* format xwd bitmaps. *Plane-major full-depth* Starbase bitmaps are translated into *XYPixmap* format xwd bitmaps.

XWD bitmaps produced by *sb2xwd* will be of the *DirectColor* visual class if the Starbase bitmap's colormap mode is *CMAP_FULL*. Other multiplane Starbase bitmaps having colormap modes of *CMAP_NORMAL* or *CMAP_MONOTONIC* will result in *PseudoColor* XWD bitmaps. Single plane Starbase bitmaps are converted to *GreyScale* XWD bitmaps.

OPTIONS

none

EXAMPLES

sb2xwd < sbfile > xwdfile
Translates the Starbase image in *sbfile* to XWD format and places the result in *xwdfile*.

sb2xwd < sbimage | xwud
Translates the image in *sbimage*, piping the results to *xwud* for display.

RESTRICTIONS

Sb2xwd accepts only *full-depth* Starbase bitmaps.

Starbase bitmaps must be 1-8, 12, or 24 planes deep. Bitmaps of depth 1-8 must have either CMAP_NORMAL or CMAP_MONOTONIC colormap modes. Bitmaps of depths 12 or 24 must have the CMAP_FULL colormap mode.

A 12 plane bitmap must be stored in three banks and have a display enable mask of 0x0F or 0xF0.

ORIGIN

Hewlett-Packard GTD

SEE ALSO

xwd(1), xwud(1), bitmapfile(4).

Starbase Graphics Techniques, HP-UX Concepts and Tutorials, chapters on "Color" and "Storing and Printing Images".

NAME

uwm - a window manager for X

SYNOPSIS

uwm [options]

DESCRIPTION

uwm is a window manager client application of the window server.

When *uwm* is invoked, it searches a predefined search path to locate any *uwm* startup files.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the *uwm* search path are:

built-in uwm startup
/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc
-f filename

To use only the settings defined in a single startup file, include the variables, **resetbindings**, **resetmenus**, **resetvariables** at the top of that specific startup file.

OPTIONS

-f *filename* Names an alternate file as a *uwm* startup file.

-display *display* Specifies the display to use; see *X(1)*.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings**, **resetmenus**, and **resetvariables** head the list.

autoselect/noautoselect
places menu cursor in first menu item. If unspecified, menu cursor is placed in the menu header when the menu is displayed.

delta=*pixels* indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)

freeze/nofreeze locks all other client applications out of the server during certain window manager tasks, such as move and resize.

grid/nogrid displays a finely-ruled grid to help you position an icon or window during resize or move operations.

hiconpad=*n* indicates the number of pixels to pad an icon horizontally. The default is five pixels.

hmenupad=*n* indicates the amount of space in pixels, that each menu item is padded to the left and right of the text.

iconfont=*fontname*
names the font that is displayed within icons. Font names for a given server can be obtained using fonts in */usr/lib/X11/fonts*.

maxcolors=*n* limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, *uwm* assumes no limit to the number of colors it can take from the color map. **maxcolors** counts colors as they are included in the file.

menufont=*fontname*
names the font that is displayed within menus. Font names for a given server can be obtained using fonts in */usr/lib/X11/fonts*.

normali/nonormali
places icons created with **f.newiconify** within the root window, even if it is placed

partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.

normalw/nonormalw
places window created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*n* moves a window *n* number of pixels or a relative amount of space, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup, f.pushdown, f.pushright**, or **f.pushleft**.

pushabsolute/pushrelative
pushabsolute indicates that the number entered with push is equivalent to pixels. When an f.push (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

pushrelative indicates that the number entered with the push variable represents a relative number. When an f.push function is called, the window is invisibly divided into the number of parts you entered with the push variable, and the window is moved one part.

resetbindings, resetmenus, and **resetvariables**
resets all previous function bindings, menus, and variables entries, specified in any startup file in the *uwm* search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname*
identifies the font of the indicator that displays in the corner of the window as you resize windows. See */usr/lib/X11/fonts* for font names.

resizerelative/noresizerelative
indicates whether or not resize operations should be done relative to moving edge or edges. By default, the dynamic rectangle uses the actual pointer location to define the new size.

reverse/noreverse
defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*n* indicates the number of pixels to pad an icon vertically. Default is five pixels.

vmenupad=*n* indicates the amount of space in pixels that the menu is padded above and below the text.

volume=*n* increases or decreases the base level volume set by the *xset(1)* command. Enter an integer from 0 to 7, 7 being the loudest.

zap/nozap causes ghost lines to follow the window or icon from its previous default location to its new location during a move or resize operation.

BINDING SYNTAX

"function=[control key(s)]:[context]:mouse events:" menu name "

Function and mouse events are required input. Menu name is required with the *f.menu* function definition only.

Function

f.beep emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup exposes the lowest window that is obscured by other windows.

f.continue releases the window server display action after you stop action with the **f.pause** function.

f.focus directs all keyboard input to the selected window. To reset the focus to all windows, invoke *f.focus* from the root window.

f.iconify when implemented from a window, this function converts the window to its respective icon. When implemented from an icon, f.iconify converts the icon to its respective window.

f.lower lowers a window that is obstructing a window below it.

f.menu invokes a menu. Enclose 'menu name' in quotes if it contains blank characters or parentheses.

f.menu=[*control key(s)*]:[*context*]:*mouse events*:"*menu name* "

f.move moves a window or icon to a new location, which becomes the default location.

f.moveopaque moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.

f.newiconify allows you to create a window or icon and then position the window or icon in a new default location on the screen.

f.pause temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.

f.pushdown moves a window down. The distance of the push is determined by the push variables.

f.pushleft moves a window to the left. The distance of the push is determined by the push variables.

f.pushright moves a window to the right. The distance of the push is determined by the push variables.

f.pushup moves a window up. The distance of the push is determined by the push variables.

f.raise raises a window that is being obstructed by a window above it.

f.refresh results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows will not refresh correctly if the exposure events are not handled properly.

f.resize resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.

f.restart causes the window manager application to restart, retracing the *uwm* search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lower case, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one, two, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager may use pointer button and control key combinations. If the window manager has bound these combinations for its own use, the client application will never see the pointer input rquested.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null). These contexts can be abbreviated as i, w, and r respectively.

The root context refers to the root, or background window, A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine

contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lower case and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down function occurs when the specified button is pressed down.

up function occurs when the specified button is released.

delta indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

```
menu = " menu name " {
"item name" : "action"
    .
    .
    .
}
```

Enter the menu name exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the f.menu function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions

Any function previously described. E.g., **f.move** or **f.iconify**.

Shell commands

Begin with an exclamation point (!) and set to run in background. You cannot include a new line character within a shell command.

Text strings

Text strings are placed in the window server's cut buffer.

Strings starting with an up arrow (^) will have a new line character appended to the string after the up arrow (^) has been stripped from it.

Strings starting with a bar character (|) will be copied as is after the bar character (|) has been stripped.

Color Menus

Use the following syntax to add color to menus:

```
menu = "menu name" (color1:color2:color3:color4) {
"item name" : (color5 :color6) : " action "
    .
    .
    .
}
```

color1 Foreground color of the header.

color2 Background color of the header.

color3 Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.

color4 Background color of the highlighter.

color5 Foreground color for the individual menu item.

color6 Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of color map entries, either before or during an invocation of *uwm*.

2) If you specify a foreground or background color that does not exist in the RGB color database of the server (see */usr/lib/X11/rgb.txt* for a sample) both the foreground and background colors default to the root window colors.

3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.

4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.

5) If you specify no colors in the startup file.

Customizing Icons

The window manager allows keyboard input in text icons. This allows the user greater flexibility towards customizing text icon names.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
freeze
grid
hiconpad=5
hmenupad=6
iconfont=fixed
menufont=fixed
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
#
#FUNCTION   KEYS  CONTEXT    BUTTON        MENU(if any)
#========   ====  =======    ======        ============
  f.menu =    meta :          : left down   :"WINDOW OPS"
  f.menu =    meta :          : middle down :"EXTENDED WINDOW OPS"
  f.move =    meta : w|i      : right down
  f.circleup = meta : root    : right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
(De)Iconify:    f.iconify
Move:           f.move
Resize:         f.resize
```

```
Lower:          f.lower
Raise:          f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:                  !"xterm &"
Iconify at New Position:        f.lowericonify
Focus Keyboard on Window:       f.focus
Freeze All Windows:             f.pause
Unfreeze All Windows:           f.continue
Circulate Windows Up:           f.circleup
Circulate Windows Down:         f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

FILES

/usr/lib/X11/rgb.txt
/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

COPYRIGHT

COPYRIGHT 1985, 1986, 1987, 1988
DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS

ALL RIGHTS RESERVED. THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION. DIGITAL MAKES NO REPRESENTATIONS ABOUT THE SUITIBILITY OF THIS SOFTWARE FOR ANY PURPOSE. IT IS SUPPLIED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY. IF THE SOFTWARE IS MODIFIED IN A MANNER CREATING DERIVATIVE COPYRIGHT RIGHTS, APPROPRIATE LEGENDS MAY BE PLACED ON THE DERIVATIVE WORK IN ADDITION TO THAT SET FORTH ABOVE. Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of Digital Equipment Corporation not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

ORIGIN

MIT Distribution

SEE ALSO

X(1), Xserver(1), xset(1)

Series 300 and 800 Only

NAME

X - a portable, network transparent window system

SYNOPSIS

X is a network transparent window system developed at MIT which runs under a wide variety of operating systems. Hewlett-Packard supports the X Window System under the Series 300 HP-UX 6.2 or higher, and the Series 800 HP-UX 3.0 or higher.

THE OFFICIAL NAMES

The official names of the software described herein are:

X
X Window System
X Version 11
X Window System, Version 11
X11

Note that the phrases X.11, X-11, X Windows or any permutation thereof, are explicitly excluded from this list and should not be used to describe the X Window System (window system should be thought of as one word).

X Window System is a trademark of the Massachusetts Institute of Technology.

DESCRIPTION

X window system servers run on computers with bitmap displays. The server distributes user input to, and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of functions, see the *Programming With Xlib* manual, the *Programming With the HP X Widgets* manual, and the *Programming With the Xt Intrinsics* manual.

The core *X* protocol provides mechanism, not policy. Windows are manipulated (including moving, resizing and iconifying) not by the server itself, but by a separate program called a "window manager" of your choosing. This program is simply another client and requires no special privileges. If you don't like the one that is supplied (see *uwm(1)*) you can write your own.

The number of programs that use *X* is growing rapidly. Of particular interest are: two terminal emulators (*hpterm(1)* and *xterm(1)*), a window manager (*uwm(1)*), a bitmap editor (*bitmap(1)*), an access control program (*xhost(1)*), user preference setting programs (*xset(1)*, *xsetroot(1)*, and *xmodmap(1)*), a load monitor (*xload(1)*), clock (*xclock(1)*), a font displayer (*xfd(1)*), a protocol translator for running X10 programs (*x10tox11(1)*), and various demos.

DISPLAY SPECIFICATION

When you first start the window system, the environment variable DISPLAY will be set (if it hasn't already been set to something) to local:0.0, in order to take advantage of local interprocess communication (IPC) mechanisms. By convention, servers on a particular machine are numbered starting with zero. The following connection protocols are supported:

TCP/IP

DISPLAY should be set to "*host*:*display*.*screen*" where *host* is the symbolic name of the machine (e.g. expo), *display* is the number of the display (usually 0), and *screen* is the number of the screen. The *screen* and preceding period are optional, with the default value being zero (0). Full Internet domain names (e.g. expo.lcs.mit.edu) are allowed for the host name.

IPC Mechanisms

DISPLAY should be set to "local:*display*.*screen*", where *display* is the display number and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

Most programs accept a command line argument of the form "-display *display*" that can be used to override the DISPLAY environment variable.

GEOMETRY SPECIFICATION

One of the advantages of using window systems over hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running, most applications accept a command line argument that is treated as the preferred size and location for this particular application's window.

This argument, usually specified as "-geometry WxH+X+Y," indicates that the window should have a width of W and height of H (usually measured in pixels or characters, depending on the application), and the upper left corner X pixels to the right and Y pixels below the upper left corner of the screen (origin (0,0)). "WxH" can be omitted to obtain the default application size, or "+X+Y" can be omitted to obtain the default application position (which is usually then left up to the window manager or user to choose). The X and Y values may be negative to position the window off the screen. In addition, if minus signs are used instead of plus signs (e.g. WxH-X-Y), then (X,Y) represents the location of the lower right hand corner of the window relative to the lower right hand corner of the screen.

By combining plus and minus signs, the window may be placed relative to any of the four corners of the screen. For example:

555x333+11+22
This will request a window 555 pixels wide and 333 pixels tall, with the upper left corner located at (11,22).

300x200-0+0
This will request a window measuring 300 by 200 pixels in the upper right hand corner of the screen.

48x48-5-10
This will request a window measuring 48 by 48 pixels whose lower right hand corner is 5 pixels off the right edge of the screen and 10 pixels off the bottom edge.

COMMAND LINE ARGUMENTS

Most X programs attempt to use a common set of names for their command line arguments. The X Toolkit automatically handles the following arguments:

-bg *color*, **-background** *color*
Either option specifies the color to use for the window background.

-bd *color*, **-bordercolor** *color*
Either option specifies the color to use for the window border.

-bw *number*, **-borderwidth** *number*
Either option specifies the width in pixels of the window border.

-display *display*
This option specifies the name of the X server to use.

-fg *color*, **-foreground** *color*
Either option specifies the color to use for text or graphics.

-fn *font*, **-font** *font*
Either option specifies the font to use for displaying text. The font is in the directory specified by xset -fp.

-geometry *geometry*
This option specifies the initial size and location of the window.

-iconic
This option indicates that application should start out in an iconic state. Note that how this state is represented is controlled by the window manager that the user is running.

-name
This option specifies the name under which resources for the application should be

found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.

-rv, -reverse
Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv
This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

-synchronous
This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

-title *string*
This option specifies the title to be used for this window. This information is used by some window managers to provide some sort of header identifying the window.

-xrm *resourcestring*
This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings of the form *"name*subname*subsubname...: value"* (see the *Xlib* manual section *Using the Resource Manager* for more details) that are loaded into a client when it starts up. The *Xlib* routine *XGetDefault(3X)* and the resource utilities within the X Toolkit obtain resources from the following sources:

RESOURCE_MANAGER root window property
Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window.

application-specific directory
Any application- or machine-specific resources can be stored in the class resource files located in the /usr/lib/X11/app-defaults directory.

$XENVIRONMENT
Any user- and machine-specific resources may be specified by setting the $XENVIRONMENT environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, the X Toolkit looks for a file named .Xdefaults-*hostname*, where *hostname* is the name of the host where the application is executing.

-xrm *resourcestring*
Applications that use the X Toolkit can have resources specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **-xrm** arguments may be given on the command line.

Program resources are organized into groups called "classes," so that collections of individual "instance" resources can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit will have at least the following resources:

background (class **Background**)
This resource specifies the color to use for the window background.

borderWidth (class **BorderWidth**)
This resource specifies the width in pixels of the window border.

borderColor (class **BorderColor**)
This resource specifies the color to use for the window border.

Most X Toolkit applications also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set Background and Foreground classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources.

When a named resource is unavailable (for example, a color named chartreuse or a font named teeneyweeney), normally no error message will be printed; whether or not useful results ensue is dependent on the particular application. If you wish to see error messages (for example, if an application is failing for an unknown reason), you may specify the value "on" for the resource named "StringConversionWarnings." If you want such warnings for all applications, specify "*StringConversionWarnings:on" to the resource manager. If you want warnings only for a single application named "zowie", specify "zowie*StringConversionWarnings:on" to the resource manager.

The available colors are found in the file /usr/lib/X11/rgb.txt. See *rgb(1)* for information on creating a new color database.

DIAGNOSTICS

The default error handler uses the Resource Manager to build diagnostic messages when error conditions arise. The default error database is stored in the file XErrorDB in the /usr/lib/X11 directory. If this file is not installed, error messages will tend to be somewhat cryptic.

COPYRIGHT

The following copyright and permission notice outlines the rights and restrictions covering most parts of the standard distribution of the X Window System from MIT. Other parts have additional or different copyrights and permissions; see the individual source files.

Copyright 1984, 1985, 1986, 1987, 1988, Massachusetts Institute of Technology.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the name of M.I.T. not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission. M.I.T. makes no representations about the suitability of this software for any purpose. It is provided "as is" without express or implied warranty.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

ORIGIN

MIT Distribution

SEE ALSO

bitmap(1), gwindstop(1), hpterm(1), init(1M), rgb(1), uwm(1), x10tox11(1), x11start(1), xclock(1), xfc(1), xfd(1), xhost(1), xinit(1), xinitcolormap(1), xload(1), xmodmap(1), xrefresh(1), xseethru(1), Xserver(1), xset(1), xsetroot(1), xterm(1), xwcreate(1), xwd(1), xwdestroy(1), xwininfo(1), xwud(1), *Programming With Xlib, Programming With the HP X Widgets and Xt Intrinsics*

Series 300 and 800 Only

NAME

x11start - start the X11 window system

SYNOPSIS

x11start [options]

DESCRIPTION

x11start starts up the X window system by running the X11 window server and selected X11 clients. By default, *hpterm(1)* and *hpwm(1)* are run by *x11start*.

x11start is a shell script that first checks the user's home directory for a *.Xdefaults* file. If there is no *.Xdefaults* file there, then the file */usr/lib/X11/sys.Xdefaults* (if it exists) will be used as a source file for *xrdb(1)*, to create a RESOURCE_MANAGER property. This is done by setting the variable $doxrdb, which is executed in the .x11start file, and thus under user control.

x11start will modify the PATH variable as needed to assure that /usr/bin/X11 is in the users PATH variable in front of /usr/bin.

x11start then runs *xinit* using the shell script *.x11start* from the user's home directory as the first argument for *xinit*. If that script does not exist or is not executable, then the script */usr/lib/X11/sys.x11start* is used as the argument for *xinit*. In any case the arguments passed to *x11start* are passed on to *xinit* following the *.x11start* argument.

FILES

/usr/lib/X11/sys.x11start
$HOME/.x11start
/usr/lib/X11/sys.Xdefaults
$HOME/.Xdefaults

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1), xinit(1), hpterm(1), hpwm(1), xrdb(1)

NAME

xclock - analog / digital clock for X

SYNOPSIS

xclock [-toolkitoptions] [-options]

DESCRIPTION

The *xclock* program displays the time in analog or digital form. The time is continuously updated at a frequency which may be specified by the user. This program is nothing more than a wrapper around the Athena Clock widget.

OPTIONS

xclock accepts all of the standard X Toolkit command line options along with the additional options listed below:

-help This option indicates that a brief summary of the allowed options should be printed on the standard error.

-analog This option indicates that a conventional 12 hour clock face with ticks marks and hands should be used. This is the default.

-digital This option indicates that a 24 hour digital clock should be used.

-chime This option indicates that the clock should chime once on the half hour and twice on the hour.

-hd *color*
This option specifies the color of the hands on an analog clock. The default is "black".

-hl *color* This option specifies the color of the edges of the hands on an analog clock, and is only useful on color displays. The default is "black".

-update *seconds*
This option specifies the frequency in seconds at which *xclock* should update its display. If the clock is obscured and then exposed, it will be updated immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

-padding *number*
This option specifies the width in pixels of the padding between the window border and clock text or picture. The default is 10 on a digital clock and 8 on an analog clock.

The following standard X Toolkit command line arguments are commonly used with *xclock:*

-bg *color*
This option specifies the color to use for the background of the window. The default is "white."

-bd *color*
This option specifies the color to use for the border of the window. The default is "black."

-bw *number*
This option specifies the width in pixels of the border surrounding the window.

-fg *color* This option specifies the color to use for displaying text on a digital clock and for the tic marks on an analog clock. The default is "black".

-fn *font* This option specifies the font to be used for digital clock text. The default is "6x10."

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the preferred size and position of the clock window.

-display *display*
This option specifies the X server to contact.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting

resources that do not have separate command line options. See *X(1)* for toolkit details.

X DEFAULTS

This program uses the *Clock* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the clock.

height (class **Height**)
Specifies the height of the clock.

update (class **Interval**)
Specifies the frequency in seconds at which the time should be redisplayed.

foreground (class **Foreground**)
Specifies the color for the tic marks. Using the class specifies the color for all things that normally would appear in the foreground color. The default is "black" since the core default for background is "white."

hands (class **Foreground**)
Specifies the color of the insides of the clock's hands.

highlight (class **Foreground**)
Specifies the color used to highlight the clock's hands.

analog (class **Boolean**)
Specifies whether or not an analog clock should be used instead of a digital one. The default is True.

chime (class **Boolean**)
Specifies whether or not a bell should be sounded on the hour and half hour.

padding (class **Margin**)
Specifies the amount of internal padding in pixels to be used. The default is 8.

font (class **Font**)
Specifies the font to be used for the digital clock. Note that variable width fonts currently will not always display correctly.

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background colors should be reversed.

ENVIRONMENT

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

NOTES

xclock believes the system clock.

Border color has to be explicitly specified when reverse video is used.

When the update is an even divisor of 60 seconds, the second hand should always be on a multiple of the update time.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1), time(3C), Athena Clock widget

Series 300 and 800 Only

NAME

xfc - font file compiler

SYNOPSIS

xfc [options]

DESCRIPTION

The **xfc** program is a bitmap distribution format to server natural format file compiler. It expects the bitmap distribution format to be in Bitmap Distribution Format 2.1. The output will be sent to standard out. If no *filename* is specified, **xfc** expects input from standard in.

OPTIONS

xfc accepts the following command line options described below.

filename

The given *filename* is a character file in Bitmap Distribution Format 2.1. See bdf(4) for more information on the format of the file.

-p*number*

Specifies that the glyphs will be padded to word boundaries. *number* can be 1,2,4 or 8.

-l Specifies that the output of the font file will be with the least significant byte first.

-m Specifies that the output of the font file will be with the most significant byte first.

ORIGIN

MIT Distribution

SEE ALSO

X(1), Xserver(1), bdf(4)

Series 300 and 800 Only

NAME

xfd - font displayer for X

SYNOPSIS

xfd [-options ...] [*fontname*]

OPTIONS

-bw *number*
Allows you to specify the width of the window border in pixels.

-rv The foreground and background colors will be switched. The default colors are black on white.

-fw Overrides a previous choice of reverse video. The foreground and background colors will not be switched.

-fg *color* On color displays, determines the foreground color (the color of the text).

-bg *color*
On color displays, determines the background color.

-bd *color*
On color displays, determines the color of the border.

-bf *fontname*
Specifies the font to be used for the messages at the bottom of the window.

-tl *title* Specifies that the title of the displayed window should be *title*.

-in *iconname*
Specifies that the name of the icon should be *iconname*.

-icon *filename*
Specifies that the bitmap in file *filename* should be used for the icon.

-verbose Specifies that verbose mode should be used.

-gray Specifies that a gray background should be used.

-start *charnum*
Specifies that character number *charnum* should be the first character displayed.

-geometry *geometry*
Specifies an initial window geometry; see *X(1)*.

-display *display*
Specifies the display to use; see *X(1)*.

DESCRIPTION

xfd creates a window in which the characters in the named font are displayed. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left will be character number 0 unless the -start option has been supplied in which case the character with the number given in the -start option will be used.

The characters are displayed in a grid of boxes, each large enough to hold any character of the font. If the -gray option has been supplied, the characters will be displayed using XDrawImageString using the foreground and background colors on a gray background. This permits determining exactly how XDrawImageString will draw any given character. If -gray has not been supplied, the characters will simply be drawn using the foreground color on the background color.

All the characters in the font may not fit in the window at once. To see additional characters, click the right mouse button on the window. This will cause the next window full of characters to be displayed. Clicking the left mouse button on the window will cause the previous window full of characters to be displayed. *xfd* will beep if an attempt is made to go back past the 0th character.

Note that if the font is a 8 bit font, the characters 256-511 (0x100-0x1ff), 512-767 (0x200-0x2ff), ... will display exactly the same as the characters 0-255 (0x00-0xff). *xfd* by default creates a window of size sufficient to display the first 256 characters using a 16 by 16 grid. In this case, there is no need to scroll forward or backward window fulls in order to see the entire contents of a 8 bit font. Of

course, this window may very well not fit on the screen...

Clicking the middle button on a character will cause that character's number to be displayed in both decimal and hexadecimal at the bottom of the window. If verbose mode is selected, additional information about that particular character will be displayed as well. The displayed information includes the width of the character, its left bearing, right bearing, ascent, and its descent. If verbose mode is selected, typing '<' or '>' into the window will display the minimum or maximum values respectively taken on by each of these fields over the entire font.

The font name is interpreted by the X server. To obtain a list of all the fonts available, examine the directory */usr/lib/X11/fonts*.

If no font name is given on the command line, *xfd* displays the font "fixed".

The window stays around until the xfd process is killed or one of 'q', 'Q', ' ', or ctrl-c is typed into the *xfd* window.

As previously mentioned, by default *xfd* attempts to create a 16 by 16 grid. This is superseded by the user's specifications. The user's specification are always used unless it is smaller than one grid. In that case, *xfd* will display one character at a time. If the user does not specify a size, then *xfd* will check to see if the message(s) it may print will fit in the space provided by the default grid. If not, it will calculate an optimum size based on its message font.

X DEFAULTS

The *xfd* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

BorderWidth
Sets the border width of the window.

BorderColor
Sets the border color of the window.

ReverseVideo
If "on", reverse the definition of foreground and background color.

Foreground
Sets the foreground color.

Background
Sets the background color.

BodyFont
Sets the font to be used in the body of the window. (I.e., for messages, etc.) This is not the font that *xfd* displays, just the font it uses to display information about the font being displayed.

IconName
Sets the name of the icon.

IconBitmap
Sets the file we should look in to get the bitmap for the icon.

Title Sets the title to be used.

ENVIRONMENT

DISPLAY
to get the default host and display to use.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANGER property.

NOTES

It should display the name of the font somewhere.

It should be rewritten to use the X toolkit.

It should skip over pages full of non-existent characters.

COPYRIGHT
Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN
MIT Distribution

SEE ALSO
X(1)

Series 300 and 800 Only

NAME

xhost - server access control program for X

SYNOPSIS

xhost [[+-]*hostname*]

DESCRIPTION

The *xhost* program is used to add and delete hosts to the list of machines that are allowed to make connections to the X server. This provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment, although it does limit the worst abuses. Environments which require more sophisticated measures should use the hooks in the protocol for passing authentication data to the server.

The server initially allows network connections only from programs running on the same machine or from machines listed in the file */etc/X*.hosts* (where * is the display number of the server). The *xhost* program is usually run either from a startup file or interactively to give access to other users.

Hostnames that are followed by two colons (::) are used in checking DECnet connections; all other hostnames are used for TCP/IP connections.

OPTIONS

Xhost accepts the following command line options described below. For security, the options that effect access control may only be run from the same machine as the server.

[+]*hostname*
: The given *hostname* (the plus sign is optional) is added to the list of machines that are allowed to connect to the X server.

-*hostname*
: The given *hostname* is removed from the list of machines that are allowed to connect to the server. Existing connections are not broken, but new connection attempts will be denied. Note that the current machine is allowed to be removed; however, further connections (including attempts to add it back) will not be permitted. Resetting the server (thereby breaking all connections) is the only way to allow local connections again.

\+
: Access is granted to everyone, even if they aren't on the list of allowed hosts (i.e. access control is turned off).

\-
: Access is restricted to only those machines on the list of allowed hosts (i.e. access control is turned on).

nothing
: If no command line arguments are given, the list of hosts that are allowed to connect is printed on the standard output along with a message indicating whether or not access control is currently enabled. This is the only option that may be used from machines other than the one on which the server is running.

FILES

/etc/X*.hosts

ENVIRONMENT

DISPLAY
: to get the default host and display to use.

NOTES

You can't specify a display on the command line because **-display** is a valid command line argument (indicating that you want to remove the machine named *"display"* from the access list).

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X*(*1*) for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1), Xserver(1)

Series 300 and 800 Only

NAME

xinit - X Window System initializer

SYNOPSIS

xinit [[client] options] [-- [server] [display] options]

DESCRIPTION

The *xinit* program is used to start the X Window System server and a first client program (usually a terminal emulator) on systems that cannot start X directly from */etc/init* or in environments that use multiple window systems. When this first client exits, *xinit* will kill the X server and then terminate.

Unless otherwise specified on the command line, *xinit* assumes that there are programs called "X" and "hpterm" in the current search path. It starts the server on display 0, sets $DISPLAY to local:0.0, and then runs an *hpterm* using the following command line:

```
xterm -geometry +1+1 -n login
```

An alternate client and/or server may be specified on the command line. The desired client program and its arguments should be given as the first command line arguments to *xinit*. To specify a particular server command line, append a double dash (--) to the xinit command line (after any client and arguments) followed by a space and the desired server command.

Both the client program name and the server program name must begin with a slash (/) or a period (.). Otherwise, they are treated as arguments to be appended to their respective startup lines. This makes it possible to add arguments (for example, foreground and background colors) without having to retype the whole command line.

If an explicit server name is not given and the first argument following the double dash (--) is a :digit, *xinit* will use that number as the display number instead of zero. All remaining arguments are appended to the server command line.

EXAMPLES

```
xinit -geometry =80x65+10+10 -fn 8x13 -j -fg white -bg navy

xinit -e widgets -- Xsun -l -c

xinit rsh fasthost cpupig -display workstation:1 -- :1 -a 2 -t 5
```

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1), Xserver(1), xterm(1)

Series 300 and 800 Only

NAME

xinitcolormap - initialize the X colormap

SYNOPSIS

xinitcolormap [options]

DESCRIPTION

This program is used to initialize the X colormap. Specific X colormap entries (pixel values) are made to correspond to specified colors. An initialized colormap is required by applications that assume a predefined colormap (e.g., many applications that use *Starbase* graphics).

xinitcolormap reads a colormap file to determine the allocation of colors in the X colormap. The name of the colormap file is determined by using (in the following order) the command line option [*-f colormapfile*], the *.Colormap* X default value or */usr/lib/X11/xcolormap*. If a colormap file is not found, then the following default colormap specification is assumed.

black (colormap entry 0)
white
red
yellow
green
cyan
blue
magenta (colormap entry 7)

xinitcolormap uses the *XStoreColor* and *XAllocColor libX11.a* calls to initialize the X colormap. The *xinitcolormap* program should be the first X client program run when the *X Window System* is started in order to assure that X colormap entries have the color associations specified in the colormap file. This could be done by running *xinitcolormap* as the first X client program in the *.x11start* file. Once *xinitcolormap* has been run, an X client program can use the initialized colors.

A colormap file is made up of lines of the form:

color

color is a one or two word color name (refer to the names in the file */usr/lib/X11/rgb.txt*), or optionally an initial sharp character followed by a numeric RGB specification (as used by the *libX.a* call *XParseColor*). The line number of a color specification in the colormap file determines the index of the color in the X colormap. Colors in the colormap file, for colormap entry 0 up to the last colormap entry to be initialized, must be specified. There should be no extra (blank or comment) lines in the colormap file. If a color is specified more than once in the colormap file then the X colormap will not be properly initialized (unless the first instance of the color is allocated in X colormap entry 0 or 1 (and is other than BlackPixel or WhitePixel) in which case the color can be specified twice).

OPTIONS

-f *colormapfile*
Specifies the file containing the colormap.

-display *display*
Specifies the server to connect to; See *X(1)* for details.

-c *count* If *count* is specified then only the first *count* colors from the colormap file will be used in initializing the X colormap.

-p If the *-p* option is specified then the colormap file will be checked for proper syntax, but the X colormap will not be initialized.

-k[ill] If the *-k*[*ill*] option is specified, then the colormap entries allocated by a previous run of *xinitcolormap* will be deallocated and the colormap will not be re-initialized. All other options will be ignored except **-display** *display*.

NOTES

xinitcolormap will only initialize the default colormap of the root window.

xinitcolormap assumes the first two colors specified are black and white.

xinitcolormap should not be run in the background. The X colormap is fully initialized only when *xinitcolormap* returns.

Running *xinitcolormap* a second time after X is started will deallocate those colors allocated by a previous run and attempt to allocate a new colormap using the new specifications. If other clients have allocated color cells that conflict with the new specifications, *xinitcolormap* will fail and the colormap will remain un-allocated.

The file */etc/newconfig/xcolormap* is a sample colormap file that corresponds to the *Starbase* default 256 entry colormap. The [*-c count*] option can be used to select a subset of the colors in this colormap file for initializing colormaps with up to 256 entries.

xinitcolormap uses XSetCloseDownMode with RetainPermanent to prevent the deallocation of the colormap. This means that *xinitcolormap* no longer spawns a daemon, and the only way for the user to be sure that *xinitcolormap* succeeded is to view the messages (or lack of) produced by *xinitcolormap*. If *x11start* is used, the output should be redirected from *xinitcolormap* to a log file.

FILES

/usr/lib/X11/xcolormap
/usr/lib/X11/rgb.txt
/etc/newconfig/xcolormap
.x11start

Series 300 and 800 Only

NAME

xload - load average display for X

SYNOPSIS

xload [*-toolkitoption* ...] [-scale *integer*] [-update *seconds*]

DESCRIPTION

The *xload* program displays a periodically updating histogram of the system load average. This program is nothing more than a wrapper around the Athena Load widget.

OPTIONS

Xload accepts all of the standard X Toolkit command line options along with the additional options listed below:

-scale *integer*
This option specifies the minimum number of tick marks in the histogram, where one division represents one load average point. If the load goes above this number, *xload* will create more divisions, but it will never use fewer than this number. The default is 1.

-update *seconds*
This option specifies the frequency in seconds at which *xload* updates its display. Expose events will cause automatic updating. The minimum as well as default time is 5 seconds.

The following standard X Toolkit arguments are commonly used with *xload*.

-bd *color*
This option specifies the border color. The default color is "black".

-bg *color*
This option specifies the background color. The default color is "white".

-bw *pixels*
This option specifies the width in pixels of the border around the window. The default value is 2.

-fg *color* This option specifies the graph color. The default color is "black".

-fn *fontname*
This option specifies the font to be used in displaying the name of the host whose load is being monitored. The default is "6x10."

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the preferred size and position of the window; see *X(1)*.

-display *display*
This option specifies the X server to contact; see *X(1)*.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Load* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the load average graph.

height (class **Height**)
Specifies the height of the load average graph.

update (class **Interval**)
Specifies the frequency in seconds at which the load should be redisplayed.

scale (class **Scale**)
Specifies the initial number of ticks on the graph. The default is 1.

minScale (class **Scale**)
Specifies the minimum number of ticks that will be displayed. The default is 1.

foreground (class **Foreground**)
Specifies the color for the graph. Using the class specifies the color for all things that normally would appear in the foreground color. The default is "black" since the core default for background is "white."

label (class **Label**)
Specifies the label to use on the graph. The default is the hostname.

font (class **Font**)
Specifies the font to be used for the label. The default is "fixed."

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

ENVIRONMENT

DISPLAY
to get the default host and display number.

XENVIRONMENT
to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

DIAGNOSTICS

Unable to open display or create window. Unable to open /dev/kmem. Unable to query window for dimensions. Various X errors.

NOTES

This program requires the ability to open and read /dev/kmem. On most systems, this requires the suid bit set with root ownership or the sgid bit set and membership in the same group as /dev/kmem.

Reading /dev/kmem is inherently non-portable.

Border color has to be explicitly specified when reverse video is used.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1), xrdb(1), mem(7), Athena Load widget

Series 300 and 800 Only

NAME

xmodmap, xprkbd - keyboard modifier utilities for X

SYNOPSIS

xmodmap [-options ...] [*filename*]

xprkbd [-display *display*]

DESCRIPTION

Xmodmap is a utility for displaying and altering the X keyboard modifier map and keysym table on the specified display and host. It is intended to be run from a user's X startup script to setup the keyboard according to personal tastes.

With no arguments, *xmodmap* displays the current map.

Xprkbd prints on the standard output a table of the keycodes, the keysym code, and the keynames for the keyboard on the appropriate X server.

OPTIONS

Both programs accept the following option:

-display *display*
This option specifies the host and display to use; see *X(1)*.

The *xmodmap* program also accepts:

-help This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xmodmap*.

-grammar
This option indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.

-verbose This option indicates that *xmodmap* should print logging information as it parses its input.

-quiet This option turns off the verbose logging. This is the default.

-n This option indicates that *xmodmap* should not change the mappings, but should display what it would do, like *make(1)* does when given this option.

-e *expression*
This option specifies an expression to be executed. Any number of expressions may be specified from the command line.

-p This option indicates that the current modifier map should be printed on the standard output.

- A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like "*.keymap.km*".

For compatibility with an older version, *xmodmap* also accepts the following obsolete single letter options:

-[SLC12345]
These options indicate that all current keys for the Shift, Lock, Control, or Mod modifier sets should be removed from the modifier map. These are equivalent to **clear** expressions.

-[slc] *keysym*
These options specify a *keysym* to be removed from the Shift, Lock, or Control modifier sets. These are equivalent to **remove** expressions.

+[slc12345] *keysym*
These options specify a *keysym* to be added to the Shift, Lock, or Control modifier sets. These are equivalent to **add** expressions.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and converts them into appropriate calls to the *Xlib* routines *XChangeKeyboardMapping*, *XInsertModifiermapEntry* and *XDeleteModifiermapEntry*. Allowable expressions include:

keycode *KEYCODE* = *KEYSYMNAME* ...
The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal). Usually only one keysym is assigned to a given code.

keysym *KEYSYMNAME* = *KEYSYMNAME* ...
The *KEYSYMNAME* on the left hand side is looked up to find its current keycode and the line is replaced with the appropriate **keycode** expression. Note that if you have the same keysym bound to multiple keys, this might not work.

clear *MODIFIERNAME*
This removes all entries in the modifier map for the given modifier, where valid name are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names, although it does matter for all other names). For example, "clear Lock" will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME* = *KEYSYMNAME* ...
This adds the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the EXAMPLES section).

remove *MODIFIERNAME* = *KEYSYMNAME* ...
This removes the given keysyms from the indicated modifier map. Unlike **add,** the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

Lines that begin with an exclamation mark (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

To make the backspace key generate a delete instead, use

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

To swap the left control and caps lock keys you could use:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

As a more complicated example, the following is what the author uses:

```
!
! On the HP, the following keycodes have key caps as listed:
!
!   101  Backspace
!    55  Caps
!    14  Ctrl
!    15  Break/Reset
!    86  Stop
!    89  F5
!
```

```
! I prefer using "keycode" over "keysym" so that I can rerun the file to
! fix up my keyboard.
!
! This sets the backspace key to generate Delete, flushes all caps lock
! bindings, assigned a control key to what used to be the caps lock key,
! makes the F1 generate ESC, and makes the Break/Reset key be a shift lock.

keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

ENVIRONMENT

DISPLAY

to get default host and display number.

NOTES

Every time a **keycode** expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
Copyright 1987 Sun Microsystems, Inc.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

xpr - print an X window dump

SYNOPSIS

xpr [**-scale** *scale*] [**-density** *dpi*] [**-height** *inches*] [**-width** *inches*] [**-left** *inches*] [**-top** *inches*] [**-header** *string*] [**-trailer** *string*] [**-landscape**] [**-portrait**] [**-rv**] [**-compact**] [**-output** *filename*] [**-append** *filename*] [**-noff**] [**-split** *n*] [**-device** *dev*] [**-cutoff** *level*] [**-noposition**] *filename*

DESCRIPTION

Xpr takes as input a window dump file produced by *xwd(1)* and formats it for output on the HP LaserJet (or other PCL printers), HP PaintJet, LN03, LA100, PostScript printers, or IBM PP3812 page printer. If no *filename* argument is given, the standard input is used. By default, *xpr* prints the largest possible representation of the window on the output page. Options allow the user to add headers and trailers, specify margins, adjust the scale and orientation, and append multiple window dumps to a single output file. Output is to standard output unless **-output** is specified.

Command Options

-scale *scale*
Affects the size of the window on the page. The HP, LN03 and PostScript printers are able to translate each bit in a window pixel map into a grid of a specified size. For example each bit might translate into a 3x3 grid. This would be specified by **-scale** *3*. By default a window is printed with the largest scale that will fit onto the page for the specified orientation.

-density *dpi*
Indicates what dot-per-inch density should be used by the printer.

-height *inches*
Specifies the maximum height of the window on the page.

-width *inches*
Specifies the maximum width of the window.

-left *inches*
Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

-top *inches*
Specifies the top margin for the picture in inches. Fractions are allowed.

-header *string*
Specifies a header string to be printed above the window.

-trailer *string*
Specifies a trailer string to be printed below the window.

-landscape
Forces the window to printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

-portrait
Forces the window to be printed in portrait mode. By default a window is printed such that its longest side follows the long side of the paper.

-rv Forces the window to be printed in reverse video.

-compact
Uses simple run-length encoding for compact representation of windows with lots of white pixels.

-output *filename*
Specifies an output file name. If this option is not specified, standard output is used.

-append *filename*
Specifies a filename previously produced by *xpr* to which the window is to be appended.

-noff When specified in conjunction with **-append**, the window will appear on the same page as the previous window.

-split *n* This option allows the user to split a window onto several pages. This might be necessary for very large windows that would otherwise cause the printer to overload and print the page in an obscure manner.

-device *dev*
Specifies the device on which the file will be printed. Currently *xpr* understands the following *dev*s:

ljet	HP LaserJet series and other monochrome PCL devices such as ThinkJet, QuietJet, RuggedWriter, HP2560 series, and HP2930 series printers
pjet	HP PaintJet (color mode)
ln03	DEC LN03
la100	DEC LA100
ps	PostScript printers
pp	IBM PP3812

The default device is *ljet*. **-device lw** (LaserWriter) is equivalent to -device ps and is provided only for backwards compatibility.

-cutoff *level*
Changes the intensity level where colors are mapped to either black or white for monochrome output on a LaserJet printer. The *level* is expressed as percentage of full brightness. Fractions are allowed.

-noposition
This option causes header, trailer, and image positioning command generation to be bypassed for LaserJet and PaintJet printers.

LIMITATIONS

The current version of *xpr* can generally print out on the LN03 most X windows that are not larger than two-thirds of the screen. For example, it will be able to print out a large Emacs window, but it will usually fail when trying to print out the entire screen. The LN03 has memory limitations that can cause it to print incorrectly very large or complex windows. The two most common errors encountered are "band too complex" and "page memory exceeded." In the first case, a window may have a particular six pixel row that contains too many changes (from black to white to black). This will cause the printer to drop part of the line and possibly parts of the rest of the page. The printer will flash the number '1' on its front panel when this problem occurs. A possible solution to this problem is to increase the scale of the picture, or to split the picture onto two or more pages. The second problem, "page memory exceeded," will occur if the picture contains too much black, or if the picture contains complex half-tones such as the background color of a display. When this problem occurs the printer will automatically split the picture into two or more pages. It may flash the number '5' on its front panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or rework to application to produce a less complex picture.

Xpr provides some support for the LA100. However, there are several limitations on its use: The picture will always be printed in portrait mode, there is no scaling and the aspect ratio will be slightly off.

Support for PostScript output currently cannot handle the **-append**, **-noff** or **-split** options.

The **-compact** option is *only* supported for PostScript output. It compresses white space but not black space, so it is not useful for reverse-video windows.

HP PRINTERS

If no **-density** is specified on the command line 300 dots per inch will be assumed for *ljet* and 90

dots per inch for *pjet*. Allowable *density* values for a LaserJet printer are 300, 150, 100, and 75 dots per inch. Consult the operator's manual to determine densities supported by other printers.

If no **-scale** is specified the image will be expanded to fit the printable page area.

The default printable page area is 8x10.5 inches. Other paper sizes can be accomodated using the **-height** and **-width** options.

Note that a 1024x768 image fits the default printable area when processed at 100 dpi with scale=1, the same image can also be printed using 300 dpi with scale=3 but will require considerably more data be transfered to the printer.

Xpr may be tailored for use with monochrome PCL printers other than the LaserJet. To print on a ThinkJet (HP2225A) *xpr* could be invoked as:

xpr -density 96 -width 6.667 *filename*

or for black-and-white output to a PaintJet:

xpr -density 180 *filename*

The monochrome intensity of a pixel is computed as 0.30*R + 0.59*G + 0.11*B. If a pixel's computed intensity is less than the **-cutoff** level it will print as white. This maps light-on-dark display images to black-on-white hardcopy. The default cutoff intensity is 50% of full brightness. Example: specifying **-cutoff 87.5** moves the white/black intensity point to 87.5% of full brightness.

A LaserJet printer must be configured with sufficient memory to handle the image. For a full page at 300 dots per inch approximately 2MB of printer memory is required.

Color images are produced on the PaintJet at 90 dots per inch. The PaintJet is limited to sixteen colors from its 330 color palette on each horizontal print line. *Xpr* will issue a warning message if more than sixteen colors are encountered on a line. *Xpr* will program the PaintJet for the first sixteen colors encountered on each line and use the nearest matching programmed value for other colors present on the line.

Specifying the **-rv**, reverse video, option for the PaintJet will cause black and white to be interchanged on the output image. No other colors are changed.

Multiplane images must be recorded by *xwd* in *ZPixmap* format. Single plane (monochrome) images may be in either *XYPixmap* or *ZPixmap* format.

Some PCL printers do not recognize image positioning commands. Output for these printers will not be centered on the page and header and trailer strings may not appear where expected.

The **-split** option is not supported for HP printers.

COPYRIGHT

Copyright 1988, Hewlett Packard Company.
Copyright 1988, Massachusetts Institute of Technology.
Copyright 1986, Marvin Solomon and the University of Wisconsin.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

xwd(1), xdpr(1), xwud(1), X(1)

NAME

xrdb - X server resource database utility

SYNOPSIS

xrdb [-option ...] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file. The resource manager (used by the Xlib routine *XGetDefault(3X)* and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editting files. For compatibility, if there is no RESOURCE_MANAGER property defined (either because xrdb was not run or if the property was removed), the resource manager will look for a file called *.Xdefaults* in your home directory. The *filename* (or the standard input if - or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

HOST=hostname
the hostname portion of the display to which you are connected.

WIDTH=num
the width of the screen in pixels.

HEIGHT=num
the height of the screen in pixels.

X_RESOLUTION=num
the x resolution of the screen in pixels per meter.

Y_RESOLUTION=num
the y resolution of the screen in pixels per meter.

PLANES=num
the number of bit planes for the default visual.

BITS_PER_RGB=num
the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it is not related to the number of planes, which the log base 2 of the size of the colormap.

CLASS=visualclass
one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor.

COLOR only defined if the default visual's type is one of the color options. Lines that begin with an exclamation mark (!) are ignored and may be used as comments.

OPTIONS

xrdb program accepts the following options:

-help This option (or any unsupported option) will cause a brief description of the allowable options and parameters to be printed.

-display *display*
This option specifies the X server to be used; see *X(1)*.

-cpp *filename*
This option specifies the pathname of the C preprocessor program to be used. Although *xrdb* was designed to use CPP, any program that acts as a filter and accepts the -D, -I, and -U options may be used.

-nocpp This option indicates that *xrdb* should not run the input file through a preprocessor before loading it into the RESOURCE_MANAGER property.

-symbols
This option indicates that the symbols that are defined for the preprocessor should be

printed onto the standard output. It can be used in conjunction with **-query,** but not with the options that change the RESOURCE_MANAGER property.

-query This option indicates that the current contents of the RESOURCE_MANAGER property should be printed onto the standard output. Note that since preprocessor commands in the input resource file are part of the input file, not part of the property, they won't appear in the output from this option. The **-edit** option can be used to merge the contents of the property back into the input resource file without damaging preprocessor commands.

-load This option indicates that the input should be loaded as the new value of the RESOURCE_MANAGER property, replacing whatever what there (i.e. the old contents are removed). This is the default action.

-merge This option indicates that the input should be merged with, instead of replacing, the current contents of the RESOURCE_MANAGER property. Since *xrdb* can read the standard input, this option can be used to the change the contents of the RESOURCE_MANAGER property directly from a terminal or from a shell script.

-remove This option indicates that the RESOURCE_MANAGER property should be removed from its window.

-edit *filename*
This option indicates that the contents of the RESOURCE_MANAGER property should be edited into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

-backup *string*
This option specifies a suffix to be appended to the filename used with **-edit** to generate a backup file.

-D*name*[=*value*]
This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef.*

-U*name* This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

-I*directory*
This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include.*

FILES

Generalizes ~/*.Xdefaults* files.

ENVIRONMENT

DISPLAY
to figure out which display to use.

NOTES

The default for no arguments should be to query, not to overwrite, so that it is consistent with other programs.

COPYRIGHT

Copyright 1988, Digital Equipment Corporation.

ORIGIN

MIT Distribution

SEE ALSO

X(1), XGetDefault(3X), Xlib Resource Manager documentation

Series 300 and 800 Only

NAME

xrefresh - refresh all or part of an X screen

SYNOPSIS

xrefresh [-options]

DESCRIPTION

xrefresh is a simple X program that causes all or part of your screen to be repainted. *xrefresh* maps a window on top of the desired area of the screen and then immediately unmaps it, causing refresh events to be sent to all applications. By default, a window with no background is used, causing all applications to repaint "smoothly." However, the various options can be used to indicate that a solid background (of any color) or the root window background should be used instead.

OPTIONS

-white Use a white background. The screen just appears to flash quickly, and then repaint.

-black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.

-solid *color*
Use a solid background of the specified color.

-root Use the root window background.

-none This is the default. All of the windows simply repaint.

-geometry *geometry*
Specifies the portion of the screen to be repainted; see *X(1)*.

-display *display*
This argument allows you to specify the server and screen to refresh; see *X(1)*.

X DEFAULTS

The *xrefresh* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

Black Uses black for the window background. Default is False.

White Uses white for the window background. Default is False.

Solid Uses the given color for the window background.

None Maps an invisible window to the screen. Default is True.

Root Uses the root window background for the window background. Default is False.

Geometry
Determines the area to refresh.

ENVIRONMENT

DISPLAY - To get default host and display number.

NOTES

It should have just one default type for the background.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

xseethru - X Window System, opens a transparent window into the HP 98550A, HP 98720A or HP 98730A Graphics Display Systems overlay plane.

SYNOPSIS

xseethru [options]

DESCRIPTION

xseethru opens a transparent window into the HP 98550A, HP 98720A or HP 98730A Graphics Display System overlay plane. It's used typically to view Starbase graphics in the image planes while running X in the overlay planes.

OPTIONS

-geometry *geometry*

The transparent window is created with the specified size according to the geometry specification. See *X(1)* for details.

-display *display*

Specified the sever to use; see *X(1)* for details.

ENVIRONMENT

DISPLAY

To get the default host and display number.

HARDWARE DEPENDENCIES

Only one display: 0, is supported on the HP 9000 Series 300.
xseethru is only useful on the HP 98550A, HP 98720A or HP 98730A Graphics Display System.

ORIGIN

Hewlett-Packard Company

SEE ALSO

X(1)

Series 300 and 800 Only

NAME

X - X Window System server

SYNOPSIS

X *:displaynumber* [-option] *ttyname*

DESCRIPTION

X is the window system server. It is started from the *xinit(1)* program, which is called by *x11start*. The *displaynumber* argument is used by clients in their DISPLAY environment variables to indicate which server to contact (large machines may have several displays attached). This number can be any number, but there can't be more than 4 of them. If no number is specified 0 is used. This number is also used in determining the names of various startup files. The *ttyname* argument is passed in by *init* and isn't used.

The executable that is invoked when *X* is run is actually one of a collection of programs that depend on the hardware that is installed on the machine. Any additional features are described in the documentation for that server.

The Hewlett-Packard server has support for the following protocols:

TCP/IP

The server listens on port htons(6000+N), where N is the display number.

Local IPC Mechanism

The file name for the socket is /usr/spool/sockets/X11/* where "*" is the display number.

When the server starts up, it takes over the display. If you are running on a workstation whose console is the display, you cannot log into the console while the server is running.

OPTIONS

The following options can be given on the command line to any X server, usually when it is started by *init(1M)*.

-a *number*

sets pointer acceleration (i.e. the ratio of how much is reported to how much the user actually moved the pointer).

-c turns off key-click.

c *volume*

sets key-click volume (allowable range: 0-100). Default is 50.

-f *volume*

sets feep (bell) volume (allowable range: 0-100). Default is 50.

-logo turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client. Default is -logo. This must be used in conjuction with -v.

nologo turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.

-p *minutes*

sets screen-saver pattern cycle time in minutes. Default is 10 minutes.

-r turns off auto-repeat.

r turns on auto-repeat.

-s *minutes*

sets screen-saver timeout time in minutes. Default is 10 minutes.

-t *numbers*

sets pointer acceleration threshold in pixels (i.e. after how many pixels pointer acceleration should take effect).

-to *seconds*

sets connection timeout in seconds. Default is 60 seconds.

v sets video-on screen-saver preference.

-v sets video-off screen-saver preference

-co *filename*
sets name of RGB color database

-help prints a usage message

-fp *fontPath*
sets the search path for fonts

-fc *cursorFont*
sets default cursor font

-fn *font* sets the default font

RUNNING FROM INIT

To run X from *init*, it is necessary to modify */etc/inittab* and */etc/gettydefs*. Detailed information on these files may be obtained from the *inittab*(*4*) and *gettydefs*(*4*) man pages.

To run X from *init* on display 0, with a login *xterm* running on */dev/ttypf*, in init state 3, the following line must be added to */etc/inittab*:

X0:3:respawn:env PATH=/bin:/usr/bin/X11:/usr/bin xinit -L ttyqf -- :0

To run X with a login *hpterm*, the following should be used instead:

X0:3:respawn:env PATH=/bin:/usr/bin/X11:/usr/bin xinit hpterm =+1+1 -n login -L ttyqf -- :0

In addition, the following line must be added to */etc/gettydefs* (this should be a single line):

Xwindow# B9600 HUPCL PARENB CS7 # B9600 SANE PARENB CS7 ISTRIP IXANY TAB3 #X login: #Xwindow

There should not be a getty running against the display for states in which X is run from *xinit*.

SECURITY

X uses an access control list for deciding whether or not to accept a connection from a given client. This list initially consists of the machine on which the server is running, and any hosts listed in the file */etc/X*.hosts* (where * is the display number). This file should contain one line per host name, with no white space.

The user can manipulate a dynamic form of this list in the server using the *xhost*(*1*) program from the same machine as the server.

Unlike some window systems, *X* does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen.

SIGNALS

X will catch the SIGHUP signal sent by *init*(*1M*) after the initial process (usually the login terminal window) started on the display terminates. This signal causes all connections to be closed (thereby "disowning" the terminal), all resources to be freed, and all defaults restored.

DIAGNOSTICS

Too numerous to list them all. If run from *init*(*1M*), errors are logged in the file */usr/adm/X*msgs*,

FILES

/etc/inittab	Script for the init process
/etc/gettydefs	Speed and terminal settings used by getty
/etc/X*.hosts	Initial access control list
/usr/lib/X11/fonts	Font directory
/usr/lib/X11/rgb.txt	Color database
/usr/lib/X11/rgb.pag	Color database

/usr/lib/X11/rgb.dir	Color database
/usr/spool/sockets/X11/*	IPC mechanism socket
/usr/adm/X*msgs	Error log file
/usr/lib/X11/X*devices	Input devices used by the server
/usr/lib/X11/X*screens	Screens used by the server
/usr/lib/X11/X*pointerkeys	Keyboard pointer device file

NOTES

The option syntax is inconsistent with itself and *xset(1)*.

The acceleration option should take a numerator and a denominator like the protocol.

If *X* dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The color database is missing a large number of colors. However, there doesn't seem to be a better one available that can generate RGB values.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

bitmap(1), getty(1M), gettydefs(4), gwindstop(1), hpterm(1), init(1M), inittab(4), rgb(1), uwm(1), x10tox11(1), x11start(1), xclock(1), xfc(1), xfd(1), xhost(1), xinit(1), xinitcolormap(1), xload(1), xmodmap(1), xrefresh(1), xseethru(1), Xserver(1), xset(1), xsetroot(1), xterm(1), xwcreate(1), xwd(1), xwdestroy(1), xwininfo(1), xwud(1), *Programming With Xlib, Programming With the HP X Widgets and Xt Intrinsics*

Series 300 and 800 Only

NAME

xset - user preference utility for X

SYNOPSIS

xset [options]

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS

-display *display*

This option specifies the server to use; see *X(1)*.

b the **b** option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

c The **c** option controls key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, keyclick will be disabled. If a value from 0 to 100 is given, it is used to indicate volume, as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp [*path...*]

The **fp** option sets the font path. It may be followed by a comma-separated list of directories or the flag 'default'. The indicated path will be used to find fonts for clients. No parameters or **fp default** will restore the default font path.

-fp or **fp-** *path*[,*path...*]

The **-fp** and **fp-** options remove elements from the current font path. They must be followed by a comma-separated list of directories, each ending with '/'.

+fp or **fp+** *path*[,*path...*]

The **+fp** and **fp+** options prepend and append elements to the current font path, respectively. They must be followed by a comma-separated list of directories.

led The **led** option controls the keyboard LEDs. This controls the turning on or off of one or all of the LEDs. It accepts an optional integer, a preceding dash(-) or an 'on/off' flag. If no parameter or the 'on' flag is given, all LEDs are turned on. If a preceding dash or the flag 'off' is given, all LEDs are turned off. If a value between 1 and 32 is given, that LED will be turned on or off depending on the existence of a preceding dash. A common LED which can be controlled is the "Caps Lock" LED. "xset led 3" would turn led #3 on. "xset -led 3" would turn it off. The particular LED values may refer to different LEDs on different hardware.

m [*acceleration* [*threshold*]]

The **m** option controls the mouse parameters. The parameters for the mouse are 'acceleration' and 'threshold'. The mouse, or whatever pointer the machine is connected to, will go 'acceleration' times as fast when it travels more than 'threshold' pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen in a flick of the wrist when desired. One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.

p *pixel color*

The **p** option controls pixel color values. The parameters are the color map entry number in decimal, and a color specification. The root background colors may be

changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and 1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.

pm The **pm** option controls pointer map entries. The parameters are pointer button codes or the flag 'default'. The default is 1, 2, 3, ... up to the currently-defined number of buttons on the pointer. If no parameters or the flag 'default' is used, the system defaults will be set.

r The **r** option controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.

s The **s** option lets you set the screen saver parameters. This option accepts up to two numerical parameters, a 'blank/noblank' flag, an 'expose/noexpose' flag, an 'on/off' flag, or the 'default' flag. If no parameters or the 'default' flag is used, the system will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.

q The **q** option gives you information on the current settings.

These settings will be reset to default values when you log out.

HARDWARE DEPENDENCIES

Note that not all X implementations are guaranteed to honor all of these options.

series 800

There is no hardware support for changing bell pitch or duration.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1), Xserver(1), xmodmap(1), xsetroot(1)

Series 300 and 800 Only

NAME

xsetroot - root window parameter setting utility for X

SYNOPSIS

xsetroot [options]

DESCRIPTION

The *setroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if *-def* is specified, the window is reset to its default state. the *-def* option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (-solid, -gray, -grey, -bitmap, and -mod) may be specified at a time.

OPTIONS

The various options are as follows:

-help Print a usage message and exit.

-def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*
This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be created with the *bitmap(1)* program. You probably want the mask file to be all black until you get used to the way masks work.

-bitmap *filename*
Use the bitmap specified in the file to set the window pattern. You can create your own bitmap files using the *bitmap(1)* program. The entire background will be made up of repeated "tiles" of the bitmap.

-mod *x y*
This is used to create a plaid-like grid pattern on your screen. x and y are integers ranging from 1 to 16. Zero and negative numbers are taken as 1.

-gray Make the entire background gray. (Easier on the eyes.)

-grey Make the entire background grey.

-fg *color*
Use "color" as the foreground color when setting attributes. Options that use/are affected by this parameter are -bitmap, -cursor, -mod, -gray and -grey.

-bg *color*
Use "color" as the background color when setting attributes. Options that use/are affected by this parameter are -bitmap, -cursor, -mod, -gray and -grey.

-rv This exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.

-solid *color*
Set the window color to "color".

-name *string*
Set the name of the root window to "string". There is no default value. Usually a name is assigned to a window so that the window manager can use a text representation when the window is iconified. This option is unused since you can't iconify the background.

-display *display*
Specifies the server to connect to; see *X(1)*.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN
MIT Distribution

SEE ALSO
X(1), xset(1)

Series 300 and 800 Only

NAME

xterm - terminal emulator for X

SYNOPSIS

xterm [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *xterm* program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), *xterm* will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be choosen through escape sequences, the "Modes" menu in the VT102 window, and the "Tektronix" menu in the 4014 window.

OPTIONS

The *xterm* terminal emulator accepts all of the standard X Toolkit command line options along with the additional options listed below (if the option begins with a '+' instead of a '-', the option is restored to its default value):

-132 Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the *xterm* window will resize appropriately.

-ah This option indicates that *xterm* should always highlight the text cursor and borders. By default, *xterm* will display a hollow text cursor whenever the focus is lost or the pointer leaves the window.

+ah This option indicates that *xterm* should do text cursor highlighting.

-b *number*
This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.

-cc *characterclassrange:value*[**,...**]
This sets classes indicated by the given ranges for using in selecting by words. See the section specifying character classes.

-cr *color* This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

-cu This option indicates that *xterm* should work around a bug in the *curses*(3x) cursor motion package that causes the *more*(1) program to display lines that are exactly the width of the window and are followed by line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu This option indicates that that *xterm* should not work around the *curses*(3x) bug mentioned above.

-e *program* [*arguments* ...]
This option specifies the program (and its command line arguments) to be run in the *xterm* window. It also sets the window title and icon name to be the basename of the program being executed if neither *-T* nor *-n* are given on the command line. **This must be the last option on the command line.**

-fb *font* This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by

overstriking this font. The default bold font is "vtbold." The directory containing the font is determined by xset -fp.

-j This option indicates that *xterm* should do jump scrolling. Normally, text is scrolled one line at a time; this option allows *xterm* to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.

+j This option indicates that *xterm* should not do jump scrolling.

-l This option indicates that *xterm* should send all terminal output to a log file as well as to the screen. This option can be enabled or disabled using the "xterm X11" menu.

+l This option indicates that *xterm* should not do logging.

-lf *filename*
This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (|), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*) and is created in the directory from which *xterm* was started (or the user's home directory in the case of a login window.

-ls This option indicates that shell that is started in the *xterm* window be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

+ls This option indicates that the shell that is started should not be a login shell (i.e. it will be normal "subshell").

-mb This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "Modes" menu.

+mb This option indicates that margin bell should not be rung.

-ms *color*
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.

-nb *number*
This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.

-rw This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines and is encouraged. This option can be turned on and off from the "Modes" menu.

+rw This option indicates that reverse-wraparound should not be allowed.

-s This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.

+s This option indicates that *xterm* should scroll synchronously.

-sb This option indicates that some number of lines that are scrolled off the top of the window should be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the "Modes" menu.

+sb This option indicates that a scrollbar should not be displayed.

-sf This option indicates that Sun Function Key escape codes should be generated for function keys.

+sf This option indicates that the standard escape codes should be generated for function keys.

-si This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the "Modes" menu.

+si This option indicates that output to a window should cause it to scroll to the bottom.

-sk This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal postion at the bottom of the scroll region.

+sk This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.

-sl *number*
This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.

-t This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the "Modes" menus.

+t This option indicates that *xterm* should start in VT102 mode.

-vb This option indicates that a visual bell is prefered over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.

+vb This option indicates that a visual bell should not be used.

-C This option indicates that this window should be receive console output. This is not supported on all systems.

-L This option indicates that *xterm* was started by *init*. In this mode, *xterm* does not try to allocate a new pseudoterminal as *init* has already done so. In addition, the system program *getty* is run instead of the user's shell. **This option should never be used by users when starting terminal windows.**

-S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode. This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

%geom This option specifies the prefered size and position of the Tektronix window. It is shorthand for specifying the "******tekGeometry*" resource.

#geom This option specifies the prefered position of the icon window. It is shorthand for specifying the "******iconGeometry*" resource.

-T *string* This option specifies the title for *xterm*'s windows. It is equivalent to **-title**.

-n*string* This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "******iconName*" resource.

-r This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.

-w *number*
This option specifies the width in pixels of the border surrounding the window. It is equivalent to **-borderwidth** or **-bw**.

The following standard X Toolkit command line arguments are commonly used with *xterm*:

-bg *color*
This option specifies the color to use for the background of the window. The default is "white."

-bd *color*
This option specifies the color to use for the border of the window. The default is "black."

-bw *number*
This option specifies the width in pixels of the border surrounding the window.

-fg *color* This option specifies the color to use for displaying text. The default is "black".

-fn *font* This option specifies the font to be used for displaying normal text. The default is "vtsingle." The directory containing the font is determined by xset -fp.

-name *name*
This option specifies the application name under which resource are to be obtained, rather than the default executable file name.

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry *geometry*
This option specifies the prefered size and position of the VT102 window; see *X(1)*;

-display *display*
This option specifies the X server to contact; see *X(1)*.

-xrm *resourcestring*
This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

-iconic This option indicates that *xterm* should ask the window manager to start it as an icon rather than as the normal window.

X DEFAULTS

The program understands all of the core X Toolkit resource names and classes as well as:

name (class **Name)**
Specifies the name of this instance of the program. The default is "xterm."

iconGeometry (class **IconGeometry)**
Specifies the prefered size and position of the application when iconified. It is not necessarily obeyed by all window managers.

title (class **Title)**
Specifies a string that may be used by the window manager when displaying this application.

utmpInhibit (class **UtmpInhibit)**
Specifies whether or not *xterm* should try to record the user's terminal in */etc/utmp*.

sunFunctionKeys (class **SunFunctionKeys)**
Specifies whether or not Sun Function Key escape codes should be generated for function keys instead of standard escape sequences.

The following resources are specified as part of the "vt100" widget (class "VT100"):

alwaysHighlight (class **AlwaysHighlight)**
Specifies whether or not *xterm* should always display a highlighted text cursor. By default, a hollow text cursor is displayed whenever the pointer moves out of the window or the window loses the input focus.

font (class **Font)**
Specifies the name of the normal font. The default is "vtsingle."

boldFont (class **Font)**
Specifies the name of the bold font. The default is "vtbold."

c132 (class **C132)**
Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

charClass (class **CharClass**)
Specifies comma-separated lists of character class bindings of the form [*low*-]*high*:*value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

curses (class **Curses**)
Specifies whether or not the last column bug in cursor should be worked around. The default is "false."

background (class **Background**)
Specifies the color to use for the background of the window. The default is "white."

foreground (class **Foreground**)
Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the "text" color change color. The default is "black."

cursorColor (class **Foreground**)
Specifies the color to use for the text cursor. The default is "black."

geometry (class **Geometry**)
Specifies the prefered size and position of the VT102 window.

tekGeometry (class **Geometry**)
Specifies the prefered size and position of the Tektronix window.

internalBorder (class **BorderWidth**)
Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class **JumpScroll**)
Specifies whether or not jump scroll should be used. The default is "false".

logFile (class **Logfile**)
Specifies the name of the file to which a terminal session is logged. The default is "**XtermLog.***XXXXX*" (where *XXXXX* is the process id of *xterm*).

logging (class **Logging**)
Specifies whether or not a terminal session should be logged. The default is "false."

logInhibit (class **LogInhibit**)
Specifies whether or not terminal session logging should be inhibited. The default is "false."

loginShell (class **LoginShell**)
Specifies whether or not the shell to be run in the window should be started as a login shell. The default is "false."

marginBell (class **MarginBell**)
Specifies whether or not the bell should be run when the user types near the right margin. The default is "false."

multiScroll (class **MultiScroll**)
Specifies whether or not asynchronous scrolling is allowed. The default is "false."

nMarginBell (class **Column**)
Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.

pointerColor (class **Foreground**)
Specifies the color of the pointer. The default is "black."

pointerShape (class **Cursor**)
Specifies the name of the shape of the pointer. The default is "xterm."

reverseVideo (class **ReverseVideo**)
Specifies whether or not reverse video should be simulated. The default is "false."

reverseWrap (class **ReverseWrap**)
Specifies whether or not reverse-wraparound should be enabled. The default is "false."

saveLines (class **SaveLines**)
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class **ScrollBar**)
Specifies whether or not the scrollbar should be displayed. The default is "false."

scrollInput (class **ScrollCond**)
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class **ScrollCond**)
Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

signalInhibit (class **SignalInhibit**)
Specifies whether or not the entries in the "xterm X11" menu for sending signals to *xterm* should be disallowed. The default is "false."

tekInhibit (class **TekInhibit**)
Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekStartup (class **TekStartup**)
Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class **TiteInhibit**)
Specifies whether or not *xterm* should remove remove *ti* or *te* termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string.

visualBell (class **VisualBell**)
Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is "false."

The following resources are specified as part of the "tek4014" widget (class "Tek4014"):

width (class **Width**)
Specifies the width of the Tektronix window in pixels.

height (class **Height**)
Specifies the height of the Tektronix window in pixels.

The following resources are specified as part of the "menu" widget:

menuBorder (class **MenuBorder**)
Specifies the size in pixels of the border surrounding menus. The default is 2.

menuFont (class **Font**)
Specifies the name of the font to use for displaying menu items.

menuPad (class **MenuPad**)
Specifies the number of pixels between menu items and the menu border. The default is 3.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness**)
Specifies the width in pixels of the scrollbar.

background (class **Background**)
Specifies the color to use for the background of the scrollbar.

foreground (class **Foreground**)
Specifies the color to use for the foreground of the scrollbar. The "thumb" of the

scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the *"Xterm Control Sequences"* document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the Tektronix menu; see below). The name of the file will be "COPY*yy-MM-dd.hh:mm:ss*", where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection.

Pointer button two (usually middle) 'types' (pastes) the text from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, that is you can swap "right" and "left" everywhere in the rest of this paragraph...) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window does not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing

any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer buton is pressed, the corresponding upper case letter is sent. To distinguish a pointer button from a key, the high bit of the character is set (but this is bit is normally stripped unless the terminal mode is RAW; see *tty*(4) for details).

MENUS

Xterm has three different menus, named **xterm**, **Modes**, and **Tektronix**. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two section, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The modes section contains items that apply to both the VT102 and Tektronix windows. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The **Modes** menu sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to there initial states just after *xterm* has finish processing the command line options. The **Tektronix** menu sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

CHARACTER CLASSES

Clicking the middle mouse button twice in rapid succession will cause all characters of the same class (e.g. letters, white space, punctuation) to be selected. Since different people have different preferences for what should be selected (for example, should filenames be selected as a whole or only the separate subnames), the default mapping can be overridden through the use of the *clarClass* (class *CharClass*) resource.

This resource is simply a list of *range*:*value* pairs where the range is either a single number or *low-high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. The *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The default table is:

```
static int charClass[128] = {
/* NUL SOH STX ETX EOT ENQ ACK BEL */
   32,  1,  1,  1,  1,  1,  1,  1,
/*  BS  HT  NL  VT  NP  CR  SO  SI */
    1, 32,  1,  1,  1,  1,  1,  1,
/* DLE DC1 DC2 DC3 DC4 NAK SYN ETB */
    1,  1,  1,  1,  1,  1,  1,  1,
/* CAN  EM SUB ESC  FS  GS  RS  US */
    1,  1,  1,  1,  1,  1,  1,  1,
/*  SP   !   "   #   $   %   &   ' */
   32, 33, 34, 35, 36, 37, 38, 39,
/*   (   )   *   +   ,   -   .   / */
   40, 41, 42, 43, 44, 45, 46, 47,
/*   0   1   2   3   4   5   6   7 */
```

```
  48, 48, 48, 48, 48, 48, 48, 48,
/* 8  9  :  ;  <  =  >  ?*/
  48, 48, 58, 59, 60, 61, 62, 63,
/* @  A  B  C  D  E  F  G*/
  64, 48, 48, 48, 48, 48, 48, 48,
/* H  I  J  K  L  M  N  O*/
  48, 48, 48, 48, 48, 48, 48, 48,
/* P  Q  R  S  T  U  V  W*/
  48, 48, 48, 48, 48, 48, 48, 48,
/* X  Y  Z  [  \  ]  ^  _*/
  48, 48, 48, 91, 92, 93, 94, 48,
/* `  a  b  c  d  e  f  g*/
  96, 48, 48, 48, 48, 48, 48, 48,
/* h  i  j  k  l  m  n  o*/
  48, 48, 48, 48, 48, 48, 48, 48,
/* p  q  r  s  t  u  v  w*/
  48, 48, 48, 48, 48, 48, 48, 48,
/* x  y  z  {  |  }  ~ DEL*/
  48, 48, 48,123,124,125,126,  1};
```

For example, the string "33:48,37:48,46-47:48,64:48" indicates that the exclamation mark, percent sign, period, slash, and ampersand characters should be treated the same way as characters and numbers. This is very useful for cutting and pasting electronic mailing addresses and filenames.

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replace with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

NOTES

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

This manual page is too long. There should be a separate users manual defining all of the non-standard escape sequences.

All programs should be written to use X directly; then we could eliminate this program.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

resize(1), X(1), pty(4), tty(4)

NAME

xwcreate - create a new X window

SYNOPSIS

xwcreate [-display display] [-parent parent] [-geometry geometry] [-r] [-bg color] [-bw pixels] [-bd color] [-depth depth] [-wmdir directory] [-title] name

DESCRIPTION

This command creates a new X window and assigns it the name *window_name*. This program will also create a *pty* file of the same name as the window in the indicated directory. After the window has been created, the *pty* file may be used to specify the window that an application should use when utilizing a graphics library (e.g. Starbase or HP-GKS). A window created by *xwcreate(1)* can be destroyed by *xwdestroy(1)*. Note: The window is actually created and maintained by the daemon program, *gwind*. The *xwcreate(1)* program requests the daemon to create the window. If *gwind* is not running when *xwcreate(1)* is executed, *xwcreate(1)* will start *gwind*.

-display *display*

Specifies the server to connect to; See *X(1)* for details. A limitation in xwcreate requires that the display name must be no more than 20 characters long.

-parent *parent*

Name of the window which is to be the parent of *window_name*. If named, the parent window must have been created by a previous invocation of *xwcreate(1)* and must not have been destroyed by *xwdestroy(1)*; otherwise an error message will be generated. If parent window is not named, the RootWindow of the display and screen will be used as the parent. If specified, parent window's name must be no more than 12 characters long.

-geometry *geometry*

This option specifies the preferred size and position of the window; See *X(1)* for details.

-r Requests the X server to create backing store for the window. By default, windows are not created with backing store.

-bg *color*

This option specifies the background color. By default, background color of the window will be black.

-bw *pixels*

This option specifies the width in pixels of the window border. By default, border of the window will be 3 pixels wide.

-bd *color*

This option specifies the border color. By default, the window border will be white.

-depth *depth*

This option specifies the visual depth of the window. By default, the window will have the same depth as its parent. If the specified depth is not supported by the display, an error will be generated and the window will not be created.

-wmdir *directory*

is the name of the directory where the pty file for the window will be created. If this option is not defined, then the directory name will be computed as follows: first, the environment of the process will be searched for the variable $WMDIR. If the variable $WMDIR is defined in the environment, then it will be used as the desired directory. If the the variable $WMDIR is not defined in the environment, then the pty file will be created in the default directory "/dev/screen". If the option *-wmdir* is defined in the command line, the directory name will be obtained as follows: If the directory argument implies an absolute pathname, then it will be taken to be the desired directory. Otherwise, the directory name will be taken to be relative to the value of the environment variable $WMDIR. If $WMDIR is not defined in the environment, the directory name will be taken to be relative to */dev/screen*. Note: if $WMDIR is defined in the environment, it must represent an absolute pathname. if *-wmdir* is defined in the command line, then the implied directory must have already been created. Otherwise, an error ("Invalid

directory") will be generated.

-title *name*
is the name to be used to reference the window. The *window_name* must be no more than 12 characters long.

X DEFAULTS

xwcreate(1) uses the Xlib routine *XGetDefault(3X)* to read its Xdefaults, so its resource names are all capitalized.

Background
Specifies the window's background color.

BorderColor
Specifies the border color. This option is useful only on color displays.

BorderWidth
Specifies the border width.

Depth Specifies the visual depth of the created window.

Retained
If 'on', requests the X server to create backing store for the window.

Wmdir Specifies the default directory where the pty file will be created. See *-wmdir* above for details.

Geometry
Specifies the default positioning and/or sizing for the created window. See *X(1)* for details.

EXAMPLES

xwcreate FullView
Create a window named "FullView". Since no other argument is provided, the default geometry, border color, etc. of FullView will be taken from the RootWindow of the window's display and screen.

xwcreate HalfView -display remote_host:1.2 -parent FullView
-geometry 400x200+5+10 -r -bw 10
Create a window named "HalfView" on the display "remote_host:1.2". HalfView will be a child of the window "FullView". The upper left hand corner of HalfView will be located at coordinate 5,10 of FullView and will be 400 pixels wide and 200 pixels high. The border of HalfView will be 10 pixels wide and the border colors will be the same as FullView.

ENVIRONMENT

DISPLAY - the default host and display number.
WMDIR - the window manager directory.
/dev/screen - the default window manager directory.

DIAGNOSTICS

If the window is created successfully, *xwcreate(1)* will remain silent. Otherwise *xwcreate(1)* prints one or more error messages to standard output. For example:

No such display.
Named window exists.
Named parent window does not exist.
Couldn't communicate with gwind.

NOTES

If *-wmdir* is used or *WMDIR* environment variable is set to
other than the default, the directory used must exist on the same physical device as "/dev".

If XKillClient is used (used by some window managers) on one
of the windows created by xwcreate, all xwcreate windows (started with the same "-display" argument) will also be destroyed.

ORIGIN

HP

SEE ALSO
X(1), XOpenDisplay(3x), xwdestroy(1).

NAME

xwd - dump an image of an X window

SYNOPSIS

xwd [options]

DESCRIPTION

xwd is an X Window System window dumping utility. *xwd* allows X users to store window images in a specially formatted dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing etc.. The target window is selected by clicking the mouse in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed.

OPTIONS

-help Print out the 'Usage:' command syntax summary.

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window.

-name *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window.

-nobdrs This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

-out *filename*
This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

-xy This option applies to color displays only. It selects 'XY' format dumping instead of the default 'Z' format.

-display *display*
Specifies the server to connect to; see *X(1)*.

ENVIRONMENT

DISPLAY
To get default host and display number.

NOTES

Xwd cannot get the image of child windows with a different number of planes than the specified window.

FILES

XWDFile.h
X Window Dump File format definition file.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

xwud(1), X(1)

Series 300 and 800 Only

NAME

xwd2sb - translate xwd bitmap to Starbase bitmap format

SYNOPSIS

xwd2sb

DESCRIPTION

This command translates a bitmap file created by the *xwd(1)* X window dump utility program into a Starbase bitmap file as described in *bitmapfile(4)*. Translation is done from standard input to standard output.

Bitmaps created by *xwd* in the *XYPixmap* format are translated into *plane-major full-depth* Starbase bitmaps. *ZPixmap* format bitmaps are translated into *pixel-major* Starbase bitmaps.

Xwd format bitmaps with visual class *TrueColor* or *DirectColor* are translated into Starbase bitmaps with the colormap mode *CMAP_FULL*. Other visual classes result in Starbase bitmaps with the *CMAP_NORMAL* colormap mode.

Window borders stored by *xwd* are stripped from the image during translation.

OPTIONS

none

EXAMPLES

xwd | xwd2sb | pcltrans | lp -oraw
: Invokes *xwd* to dump the contents of a window in *ZPixmap* format, *xwd2sb* translates the window image into Starbase format, *pcltrans* prepares the image for printing, and *lp* spools the image for the printer.

xwd -xy | xwd2sb > sbimage
: Invokes *xwd* to dump the contents of a window in *XYPixmap* format and *xwd2sb* to translate the image into Starbase plane-major full-depth format. The Starbase bitmap image is placed in the *sbimage* file. (Note that *pcltrans* is unable to process plane-major full-depth images.)

xwd2sb <xwdfile >sbfile
: Translates the image in *xwdfile* to Starbase format and places the result in *sbfile*.

RESTRICTIONS

XWD bitmaps must be 1-8, 12, or 24 planes deep. Bitmaps of depth 1-8 may have a visual class of GrayScale, StaticGray, PseudoColor, or StaticColor. Bitmaps of depths 12 or 24 must be of the DirectColor or TrueColor visual class.

A 12 plane bitmap must have four bits each for red, green, and blue. A 24 plane bitmap must have eight bits each for red, green, and blue.

ORIGIN

Hewlett-Packard GTD

SEE ALSO

xwd(1), pcltrans(1), bitmapfile(4).

Starbase Graphics Techniques, HP-UX Concepts and Tutorials, chapters on "Color" and "Storing and Printing Images".

Series 300 and 800 Only

NAME

xwdestroy - destroy one or more existing windows

SYNOPSIS

xwdestroy [-wmdir *directory*] *window1 window2* ...

DESCRIPTION

If a window named in the list was created using *xwcreate(1)*, then it is destroyed, along with its children. Also the *pty* devices associated with these windows are removed. Window names may not be more than 12 characters long.

-wmdir *directory*

is the name of the directory where the pty file for the window was created. If this option is not defined, then the directory name will be computed as follows: first, the environment of the process will be searched for the variable $WMDIR. If the variable $WMDIR is defined in the environment, then it will be used as the desired directory. If the variable $WMDIR is not defined in the environment, then the pty file will be destroyed in the default directory "/dev/screen". If the option *-wmdir* is defined in the command line, the directory name will be obtained as follows: If the directory argument implies an absolute pathname, then it will be taken to be the desired directory. Otherwise, the directory name will be taken to be relative to the value of the environment variable $WMDIR. If $WMDIR is not defined in the environment, the directory name will be taken to be relative to */dev/screen*. Note: if $WMDIR is defined in the environment, it must represent an absolute pathname. If *-wmdir* is defined in the command line, then the implied directory must have already been created. Otherwise, an error ("Invalid directory") will be generated.

ENVIRONMENT

WMDIR - the window manager directory.
/dev/screen - the default window manager directory.

DIAGNOSTICS

If the windows were destroyed successfully, the program remains silent. If one or more of the windows could not be destroyed because of some error, appropriate message will be printed on standard output. For example:

Invalid directoy
Named window does not exist.

ORIGIN

Hewlett-Packard Company

SEE ALSO

XOpenDisplay(3), xwcreate(1).

Series 300 and 800 Only

NAME

xwininfo - window information utility for X

SYNOPSIS

xwininfo [options]

DESCRIPTION

Xwininfo is a utility for displaying information about windows. Depending on which options are chosen, various information is displayed. If no options are chosen, **-stats** is assumed.

The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its window id on the command line with the **-id** option. In addition, if it is easier, instead of specifying the window by its id number, the **-name** option may be used to specify which window is desired by name. There is also a special **-root** option to quickly obtain information on X's root window.

OPTIONS

-help Print out the 'Usage:' command syntax summary.

-id *id* This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.

-name *name*
This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.

-root This option specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

-int This option specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.

-tree This option causes the ids and names of the root, parent, and children windows of the selected window to be displayed.

-stats This option causes various attributes of the selected window having to do with its location and appearance to be displayed. Information displayed includes the location of the window, its width and height, its depth, border width, class, and map state.

-bits This option causes various attributes of the selected window having to do with its raw bits and how it is to be stored to be displayed. Information displayed includes the window's window and bit gravities, the window's backing store hint and backing_planes value, its backing pixel, and whether or not the window has save-under set.

-events This option causes the selected window's event masks to be displayed. Both the event mask of events wanted by some client and the event mask of events not to propagate are displayed.

-size This option causes the selected window's sizing hints to be displayed. Information displayed includes both the normal size hints and the zoom size hints of the user supplied location if any, the program supplied location if any, the user supplied size if any, the program supplied size if any, the minimum size if any, the maximum size if any, the resize increments if any, and the minimum and maximum aspect ratios if any.

-wm This option causes the selected window's window manager hints to be displayed. Information displayed may include whether or not the application accepts input, what the window's icon window # and name is, where the window's icon should go, and what the window's initial state should be.

-all This option is a quick way to ask for all information possible.

-display *display*
Specifies the server to connect to; see *X(1)*.

EXAMPLE

The following is a sample summary taken with no options specified:

```
xwininfo = = > Please select the window you wish
        = = > information on by clicking the
        = = > mouse in that window.

xwininfo = = > Window id: 0x8006b (fred)

        = = > Upper left X: 0
        = = > Upper left Y: 0
        = = > Width: 1024
        = = > Height: 864
        = = > Depth: 1
        = = > Border width: 0
        = = > Window class: InputOutput
        = = > Window Map State: IsUnviewable
```

ENVIRONMENT

DISPLAY

To get default host and display number.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

X(1)

NAME

xwud - image displayer for X

SYNOPSIS

xwud [options]

DESCRIPTION

Xwud is an X Window System window image undumping utility. *Xwud* allows X users to display window images that were saved in a specially formatted dump file. The window image will appear at the coordinates of the original window from which the dump was taken. This is a crude version of a more advanced utility that has never been written. Monochrome dump files are displayed on a color monitor in the default foreground and background colors.

OPTIONS

-help Print out a short description of the allowable options.

-in *filename*
This option allows the user to explicitly specify the input file on the command line. The default is to take input from standard in.

-inverse Applies to monochrome window dump files only. If selected, the window is undumped in reverse video. This is mainly needed because the display is 'write white', whereas dump files intended eventually to be written to a printer are generally 'write black'.

-display *display*
Specifies the server to connect to; see *X(1)*.

ENVIRONMENT

DISPLAY
To get default display.

FILES

XWDFile.h
X Window Dump File format definition file.

NOTES

Does not attempt to do color translation when the destination screen does not have a colormap exactly matching that of the original window.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

ORIGIN

MIT Distribution

SEE ALSO

xwd(1), X(1)

Glossary

application program
: A computer program that performs some useful function, such as word processing or data base management.

application server
: A computer used solely to provide processing power for application programs.

active window
: The terminal window where what you type appears. If there is no active window, what you type is lost. Only one terminal window can be active at a time.

ampersand (&)
: Placed at the end of a command to specify that the client started by the command should be started as a background process. The command can be typed after the command-line prompt or included in a file such as `.x11start` or `.hpwmrc`.

background process
: A process that doesn't require the total attention of the computer for operation. Background processing enables the operating system to execute more than one program or command at a time. As a general rule, all clients should be run as background processes.

bitmap
: Generally speaking, an array of data bits used for graphic images. Strictly speaking, a pixmap of depth one (capable of 2-color images).

bitmap device
: An output device that displays bitmaps. The CRT monitor of your system is a bitmap device.

buffer

An area used for storage.

button

A button on a mouse pointing device. Mouse buttons can be mapped to the keyboard.

button binding

Association of a mouse button operation with a window manager function. For example, pressing button 3 on a window frame displays the system menu.

click

To press *and release* a mouse button. The term comes from the fact that pressing and releasing the buttons of most mice makes a clicking sound.

client

A program written specifically for the X Window System. Some clients make their own windows. Other clients are utility programs.

combined mode

A combination of image and overlay planes in which a single display has a single screen that is a combination of the image and overlay planes.

command-line prompt

A command-line prompt shows that the computer is ready to accept your commands. Each terminal emulation window has a command-line prompt that acts just like the command-line prompt you see on the screen immediately after login. Usually the command-line prompt is either a **$** (for Bourne and Korn shells) or a **%** (for C shells), but it can be modified. One popular modification is to print the current working directory and the history stack number before the **$** or **%**. You can find the command-line prompt by pressing **Return** several times. Everytime you press **Return**, HP-UX prints the prompt.

cut buffer

A buffer (memory area) that holds text that has been deleted from a file.

depth

The number of planes in a set of planes. For example, a set of 12 image planes would have a depth of 12.

diskless cluster

The networking of several systems (SPUs) together to share a common hard disk for storage of data and programs.

display

Strictly speaking, the combination of a keyboard, mouse, and one or more screens that provide input and output services to a system. While "display" is sometimes used to mean just the CRT screen, a display, as defined here, can actually include more than one physical screen.

display server

In the X Window System, the display server is the software that controls the communication between client programs and the display (keyboard, mouse, and screen combination).

double buffering

A term describing the method used by Starbase wherein half of the color planes on a monitor are used to display to the screen and the other half are used to compute and draw the next screen display. This provides smooth motion for animation and it is faster. However, it does reduce the number of colors that are available for display on the screen at one time.

double-click

To press *and release* a mouse button twice in rapid succession.

drag

To press *and hold down* a mouse button while moving the mouse on the desktop (and the pointer on the screen). Typically, dragging is used with menu selecting, moving, and resizing operations.

file server

A computer whose primary task is to control the storage and retrieval of data from hard disks. Any number of other computers can be linked to the file server in order to use it to access data. This means that less storage space is required on the individual computer.

fonts

A font is a style of printed text characters. Times Roman is the font used for most newspaper text; Helvetica is the font used for most newspaper headlines.

foreground process

A process that has the terminal window's attention. When a program is run in a window as a foreground process (as opposed to a background process), the terminal window cannot be used for other operations until the process is terminated.

graphical user interface

A form of communication between people and computers that uses graphics-oriented software such as windows, menus, and icons, to ease the burden of the interaction.

home directory

The directory in which you are placed after you log in. Typically, this is **/users/***username*, where *username* is your login name. The home directory is where you keep all "your" files.

hotspot

The area of a graphical image used as a pointer or cursor that is defined as the "point" of the pointer or cursor.

hpterm

A type of terminal window, sometimes called a "terminal emulator program" that emulates HP2622 terminals, complete with softkeys. The **hpterm** window is the default window for your X environment.

icon

A small, graphic representation of an object on the root window (typically a terminal window). Objects can be "iconified" (turned into icons) to clear a cluttered workspace and "normalized" (returned to their original appearance) as needed. Processes executing in an object continue to execute when the object is iconified.

iconify

The act of turning a window into an icon.

image mode

The default screen mode using multiple image planes for a single screen. The number of image planes determines the variety of colors that are available to the screen.

image planes

The primary display planes on a device that supports two sets of planes. The other set of display planes is known as the overlay planes. These two sets of planes are treated as two separate screens in stacked mode and one screen in combined mode.

keyboard binding

Association of a special key press with a window manager function. For example, pressing the special keys Shift Esc displays the system menu of the active window.

label

The text part of an icon.

local access

The ability to run a program on the computer you are currently operating. This is different from remote access, where you run a program on a computer that is physically removed from the one you are operating.

local client

A local client is a program that is running on your `local` computer, the same system that is running your X server.

mask

A graphical image used in conjunction with another graphical element to hide unwanted graphical effects.

matte

A border located just inside the window between the client area and the frame. It is used to create a three-dimensional effect for the frame and window.

menu

A list of selections from which to make a choice. In a graphical user interface such as the X Window System, menus enable you to control the operation of the system.

minimize

To turn a window into an icon. The terms minimize and iconify are interchangeable.

modifier key

A key that, when pressed and held along with another key, changes the meaning of the other key. CTRL, Extend char, and Shift are examples of a modifier key.

multi-tasking

The ability to execute several programs (tasks) simultaneously on the same computer.

node

An address used by the system. For example, each device on the system has its own node. The system looks there whenever it needs to access the device. A node can also be an address on a network, the location of a system.

non-client

A program that is written to run on a terminal and so must be "fooled" by a terminal emulation window into running in the window environment.

normalize

To change an icon back into its "normal" (original) appearance. The opposite of iconify.

overlay planes

The secondary set of display planes on a device that supports two sets of planes. The other set of display planes is known as the image planes. These two sets of planes are treated as two separate screens.

parent window

A window that causes another window to appear.

pixel

Short for "picture element." The individual dots, or components, of a screen. They are arranged in rows and columns and form the images that are displayed on the screen.

pixmap

An array of data bits used for graphics images. Each pixel (picture element) in the map can be several bits deep, resulting in multi-color graphics images.

pointer

Sometimes called the "mouse cursor," the pointer shows the location of the mouse. The pointer's shape depends on its location. In the root window, the pointer is an ×. On a window frame, the pointer is an arrowhead. Inside the frame, the pointer can be an arrowhead (as when it is inside a clock or load histogram frame) or an I-beam (as when it is inside a terminal window).

press

Strictly speaking, to hold down a mouse button or a key. Note that to hold down a mouse button *and move* the mouse is called "dragging."

print server

A computer that controls spooling and other printer operations. This permits a large number of individuals to efficiently share printer resources.

remote access

The ability to run a program on a computer that is physically removed from the one you are currently operating. This is different from local access, where you run a program on the computer that you are operating.

remote client

An X program that is running on a remote system, but the output of the program can be viewed on your terminal.

remote host

A computer physically removed from your own that you can log in to. See chapter 4 for prerequisites for establishing a remote host.

resource

That which controls an element of appearance or behavior. Resources are usually named for the elements they control.

restoring

The act of changing an minimized (iconified) or maximized window back to its regular size. The terms restoring and normalizing usually interchangeable.

.rhosts

A special file used in network environments that enables a remote host to log into your local system without using a password. Obviously, this has a considerable impact on the security of your system.

root menu

The menu associated with the root window. The root menu enables you to control the behavior of your environment.

root window

The root window is what the "screen" (the flat viewing surface of the terminal) becomes when you start X. To a certain extent, you can think of the root as the screen. The root window is the backdrop of your X environment. Although you can hide the root window under terminal windows or other graphic objects, you can never position anything behind the root window. All windows and graphic objects appear "stacked" on the root window.

screen

The physical CRT (Cathode Ray Tube) that displays information from the computer.

screen dump

An operation that captures an image from your screen, saves it in a bitmap file, and enables you to send that file to a printer for hardcopy reproduction.

server

A program that controls all access to input devices (typically a mouse and a keyboard) and all access to output devices (typically a display screen). It is an interface between application programs you run on your system and the system input and output devices.

stacked mode

A combination of image and overlay planes in which a single display has two "logical" screens, one the image planes, the other the overlay

planes. Typically, the image planes are used to display graphics while the overlay planes are used to display text.

system menu
The menu that displays when you press the system menu button on the HP Window Manager window frame. Every window has a system menu that enables you to control the size, shape, and position of the window.

Term0
An HP level 0 terminal. It is a reference standard that defines basic terminal functions. For more information, see *Term0 Reference* in the HP-UX documentation set.

terminal-based program
A program (non-client) written to be run on a terminal (not in a window). Terminal-based programs must be "fooled" by terminal-emulation clients to run on the X Window System.

terminal emulator
A client program that provides a window within which you can run non-client programs. The non-client program runs just as though it were running from a real terminal rather than a window acting as a terminal.

terminal type
The type of terminal attached to your computer. HP-UX uses the terminal type to set the `TERM` environment variable so that it can communicate with the terminal correctly. The terminal type is usually set at login, but can be set afterward.

terminal window
A terminal window is a window that emulates a complete display terminal. Terminal windows are typically used to "fool" non-client programs into believing they are running in their favorite terminal—not a difficult task in most cases. When not running programs or executing operating system commands, terminal windows display the command-line prompt. Two terminal windows are supplied with X11—`hpterm`, which emulates HP terminals, and `xterm`, which emulates DEC and Tektronix terminals.

text cursor

The line-oriented cursor that appears in a terminal window after the command prompt. The term is used to distinguish the cursor used by a window from the cursor used by the mouse, the pointer.

tile

A rectangular area used to cover a surface with a pattern or visual texture. The HP Window Manager supports tiling, enabling users with limited color availability to create new color tiles blended from existing colors.

title bar

The title bar is the rectangular area between the top of the window and the window frame. The title bar contains the title of the window object, usually "Terminal Emulator" for `hpterm` windows, "xclock" for clocks, and "xload" for load histograms.

transient window

A window of short duration such as a dialog box. The window is only displayed for a short time, usually just long enough to get some direction from the user.

window

A data structure that represents all or part of the CRT display screen. It contains a two-dimensional array of 16-bit character data words, a cursor, a set of current attributes, and several flags. Visually, a window is represented as a rectangular subset of the display screen.

window-based program

A client or program written for use with the X Window System. The "opposite" of a window-based program is a terminal-based program.

window decoration

The frame and window control buttons that surround windows managed by the HP Window Manager.

window manager

The window manager controls the size, placement, and operation of windows on the root window. The window manager includes the functional window frames that surround each window object as well as a menu for the root window.

X0.hosts

A file that tells the X Window System which remote hosts can access the local server and hence the local display.

xclock

An X11 client program that displays the time, either analog (hands and dial) or digital (text read out).

xload

An X11 client program that displays the work load of the system as a histogram.

xterm

An X11 client program that displays a terminal window that emulates DEC and Tektronix terminals.

Index

Special characters

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

HP Part Number
98594-90040
Microfiche No. 98594-99040
Printed in U.S.A. E1288

98594-90640
For Internal Use Only