Using the X Window System

HP Part No. B1171-90076
Printed in U.S.A. January 1995

Edition 7
DRAFT 4/7/98 12:45

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.

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.

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
September 1989	Edition 3
February 1991	Edition 4
November 1991	Edition 5
August 1992	Edition 6
January 1995	Edition 7

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

Contents

1

Introduction

Welcome to the X Window System version 11 (X11 or X). The X Window System is a network transparent window system.

The HP Visual User Environment (HP VUE) is a graphical user interface that is based on the X Window System. The X Window System can be run alone, or as part of HP VUE. This manual covers what is needed to run the X Window System by itself, although there is a lot of valuable information for the HP VUE user as well.

Hewlett Packard's X Window System also supports HP graphics application (such as Starbase) in the X Window environment. This manual includes information about this capability.

In this chapter you'll find out how this manual is organized and some of the conventions it uses.

Who Should Read this Manual

The primary audience for this manual is system administrators for systems running the X Window System *but not* HP VUE. However, HP VUE users who want information on the font server should read chapter 6, "Using Fonts."

Since HP VUE provides other mechanisms for performing some of the actions covered in this manual, HP VUE users should first look in the *HP Visual User Environment User's Manual*.

Users running graphics applications in the X Window environment will find useful information in this manual.

How This Manual Is Organized

Chapter 1	Introduction. Gives some tips, and describes other documentation available to you.
Chapter 2	Hardware and software that are part of a typical X11 system and explains general concepts.
Chapter 3	Configuration information for default file, multiple screens, remote operation, special input devices, and Native Language support.
Chapter 4	Starting, using, and stopping X.
Chapter 5	How applications obtain resources.
Chapter 6	How and where to use different fonts.
Chapter 7	Motif Window Manager.
Chapter 8	X11 clients.
Chapter 9	Special mouse and keyboard configurations.
Chapter 10	Printing and screen dumps.
Chapter 11	X Windows and graphics applications.
Appendix A	Using the Keyboards.
Glossary	Special terms.

Conventions

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

Typographical Conventions

If you see ...	It means ...
`computer text`	This text is displayed by the computer or text that you type exactly as shown. 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 you type "hpterm -fg" followed by a color you choose.
	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. Often the term is also 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. These are the default mouse button settings and can be changed as described in chapter 9.

Mouse Buttons and Their Locations

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

Be careful of your spelling:

- Watch uppercase and lowercase letters. A file named `.xdefaults` is *not* the same file as `.Xdefaults`. Use uppercase letters where indicated and *only* where indicated.
- Don't confuse the number 1 (one) with the letter "l" (el).
- Don't confuse the "0" (zero) with the upper case "O" (oh).
- 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.

For More Information

Read these books to find out more about X Windows and HP-UX.

- *Using Your HP Workstation* (B2615-90003)
- *HP Visual User Environment User's Guide* (B1171-90079)
- *Introduction to the X Window System* by Oliver Jones. Prentice Hall, Englewood Cliffs, NJ:1989.
- *The Definitive Guides to the X Window System Volume Three: X Window System User's Guide for Version 11 Release 5* by Tim O'Reilly and Valerie Quercia. O'Reilly and Associates, Petaluma, CA:1992.

2

What is the X Window System?

This chapter describes:

- Basic concepts.
- The hardware and software of a typical system.
- Distributed computing.

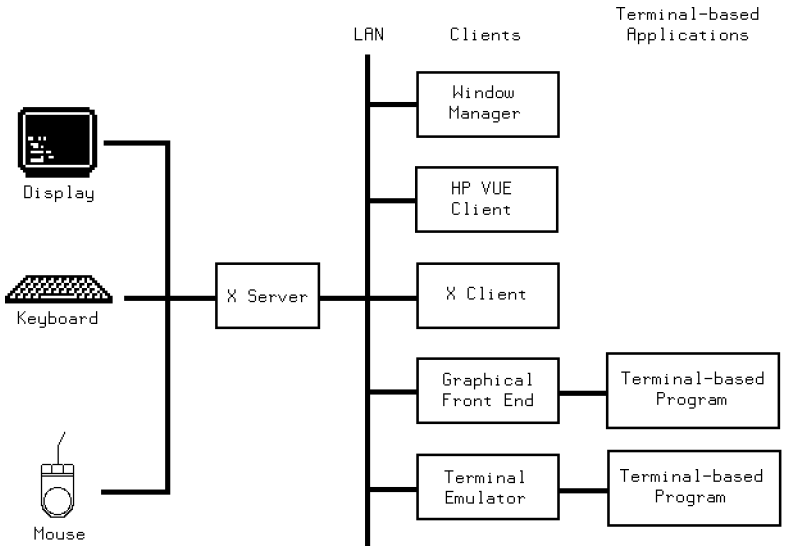
Basic Concepts

This section introduces several fundamental concepts:

- The role of the X server.
- Multi-tasking environments.
- Remote access.

The Server-Client Interaction Model

The X **server** usually starts during system boot before the login screen is displayed. The display server controls all access to input devices (typically the mouse and keyboard) and all access to display devices. You can think of it as standing between the programs running on your system and your system's input and display devices.



The Server Controls Display Access.

A **client** is any program written to run with the server. Clients know about windows and workspaces and how to make use of them. **Non-clients** are programs that don't know how to make use of windows.

Multi-Tasking

Multi-tasking is the computer's ability to execute several programs simultaneously. Each program is a separate task (process). Each process usually runs in a separate window, and processes running in separate windows do not interfere with one another. For example, you can 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.

Only one window at a time receives user input. That window is called the **active window**. While you focus on one window, other windows continue running unattended or wait for your input.

Local and Remote Access

Networked computing environments provide the ability to run programs on computers other than the one you are sitting in front of. For example, you can run a program locally and display the output on the screen of a remote system. Conversely, you can run a program remotely and display the output in a window on your screen. You can also run a program remotely and have it display on yet another remote screen.

The Parts of a Typical X11 System

All X11 systems have the following features in common:

- Computer hardware.
- The operating system.
- An X server program to control communication between the display and client programs.
- Client programs, including:
  - A window manager to control the display's window environment.
  - Application programs to provide useful services.

Hardware

The hardware system consists of several components:

System Processing Unit (SPU)

The **SPU** contains the logic circuitry that performs all the processing that takes place. The SPU runs the server, takes care of foreground and background processing, and controls local and remote accessing of your system's resources.

The Hard Disk

The hard disk stores programs and data files. Some configurations are called **diskless clusters** because groups of users share the same hard disk.

Keyboard

The keyboard is an **input device** used to type information into the computer. Although the keyboard is frequently used in conjunction with a mouse, it does not need to be. You can configure X11 so that you can use the keyboard for both text entry (its usual purpose) and for pointing and selecting (the mouse's usual purpose). **Mouseless operation** may be beneficial in situations where desk space is at a premium.

Note

There are now two keyboards available for Hewlett-Packard workstations, the 46021 keyboard, and the C1429 keyboard. See appendix B, Using the Keyboards, for more information on using these keyboards and the differences between them.

Mouse and Other Pointing Devices

A pointing device lets you point to a specific area on the screen and select it. A mouse is the most common pointing device. Mouse movements and button presses can be associated with keyboard key presses for mouseless operation.

The server also supports other pointing devices—for example a digitizer tablet or track ball. References to mouse actions apply also to corresponding actions with other devices.

Display

The display is the principal output device. A typical display consists of one physical screen per mouse and keyboard. However, a display can include as many as four physical screens, all using the same mouse and keyboard.

The screen becomes the **root window** when you boot X11. The root window contains all the windows, menus, and icons that comprise 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.

Local Area Network (LAN)

The LAN is composed of hardware and software. The hardware connects the computer system physically to a network that includes other computer systems at your site and could connect to other networks at different locations. The LAN enables you to take advantage of remote processing capabilities of X11.

Software

There are several types of software that comprise the X Window System.

To an end-user, the layers blend together into a single working environment. However, from a system administration point of view, it is important to know how the layers work together.

The Operating System

The operating system is the software that controls the operation of the computer system. The X Window System runs on the Hewlett-Packard HP-UX operating system. This is a multi-user, multi-tasking environment. A multi-user environment means more than one user can be on 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 Server

The central part of the X Window System is the **server**, also called the **X server** or **display server**. The server is the program that controls the screen, keyboard, and mouse, and processes communication requests. 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.

The Font Server

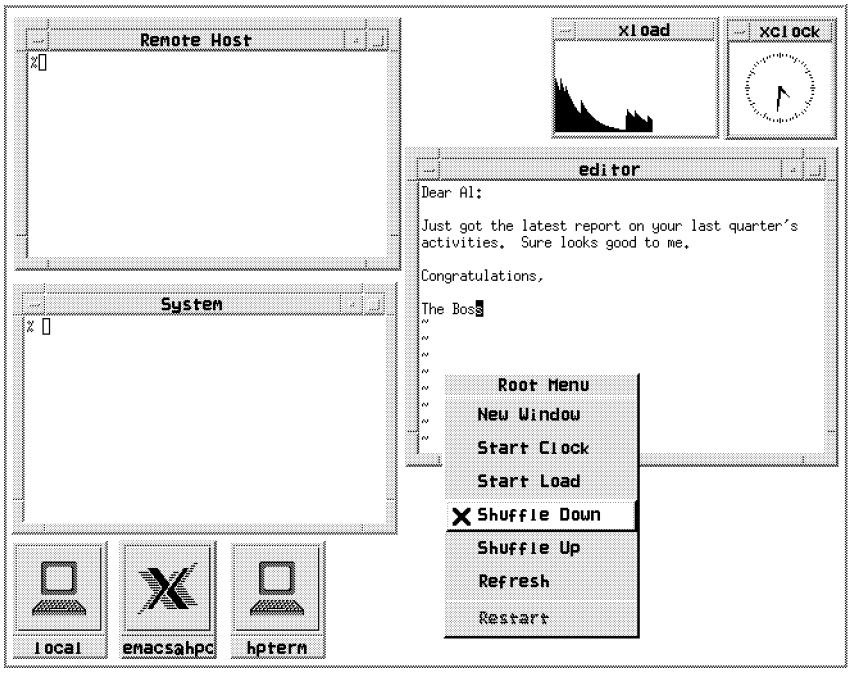
The font server allows a font administrator to distribute fonts to all X servers in a networked environment from a central point. The font server is covered in chapter 6.

The Window Manager

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. It also supplies the frames and menus for the windows.

The window manager is the first client started during a session after the X server has started. All other clients with their own windows must be able to interact with the window manager.

This manual covers the OSF/Motif Window Manager (`mwm`). Using the window manager is covered in chapter 4. Configuring the window manager is covered in chapter 7.



Windows, Clients, Menus, and Icons

X Clients

Clients are programs designed to run under the X Window System.

There are a number of clients that are included with the X Window System. For example, the `xrdb` client provides the ability to view and modify current resources.

Some clients (for example, `xwininfo` and `xmodmap`) do not create windows. They use an existing terminal emulation window to display their output.

Clients are discussed in chapter 8.

Non-Client Programs

Non-client programs are designed to run alone on display screens or "terminals" and are therefore referred to as **terminal-based** programs. Terminal-based programs must have terminal emulator windows created for them so that they can run in a window environment.

3

Preliminary Configuration

This chapter covers some of the preliminary configuration you may need to do before starting the X server. It includes:

- Setting the `DISPLAY` environment variable.
- Using hardware and software configuration files.
- Using custom screen configurations.
- Configuring the system for special input devices.
- Distributed processing.
- Using Native Language Support.

There are other chapters that deal with initial configuration for special situations:

- Chapter 4 covers starting and running X.
- Chapter 6 covers configuring and running the font server.
- Chapter 7 covers configuring the window manager.
- Chapter 11 covers configuration for running the X Window System with graphics programs, such as Starbase.

Do You Need to Read This Chapter?

All users should check:

- the `DISPLAY` variable.
- the `X0.hosts` file.
- the `/etc/hosts` file if your system is not configured to query a nameserver.

The rest of this chapter covers optional configuration. The following table shows the assumed configuration, and what you should read if you want to change it.

Default X Configuration

Expected configuration	If you want to change it, read . . .
1 display	"Using Custom Screen Configurations"
1 mouse	"Using Special Input Devices"
1 keyboard	"Using Special Input Devices"
American English language	"Customizing for Native Language Support"
X starts the `hpterm` and `mwm` clients as part of its own start-up procedures.	"Software Configuration Files" chapter 4
Default `mwm` colors, window decorations, and menus.	"Software Configuration Files" chapter 5 chapter 7
Font server not started	"Using the X11R5 Font Server" chapter 6

Finding Your System Directory

The directory containing most of the X Window System configuration files is called the *system directory*. It is `/usr/lib/X11`

Setting the DISPLAY Variable

The `DISPLAY` environment variable establishes the host, display number, and screen number to which a system sends bitmapped output.

You can check the current setting of your system's `DISPLAY` variable by typing this command:

```
env
```

A list similar to the following is displayed:

```
DISPLAY=hpaaaaa:0.0
HOME=/home/ellen
TZ=PST8PDT
⋮
```

The `DISPLAY` variable has the syntax:

$$\left[\left\{ \begin{array}{l} \textit{hostname} \\ \texttt{local} \\ \texttt{unix} \\ \texttt{shmlink} \end{array} \right\} \right] : \textit{display[.screen]}$$

The default is *hostname*`:0.0`, which is display 0, screen 0 of the display running the X server.

To reset the `DISPLAY` environment variable type the appropriate command shown below, or put it into the configuration file used by your system if you want it to be in effect every time you log in.

Setting Environment Variables

Shell	Command	Configuration File
sh	`DISPLAY=`*host:display.screen* `export DISPLAY`	`~/.profile`
csh	`setenv DISPLAY` *host:display.screen*	`~/.login`
Aegis	`DISPLAY :=` *host:display.screen* `export DISPLAY`	`~/user_data/startup_dm.`*xxx* or `/sys/dm/startup_login.`*xxx*
ksh	`DISPLAY=`*host:display.screen* `export DISPLAY`	`~/.profile`

Making an X0.hosts File

The `/etc/X0.hosts` file is an ASCII text file containing the hostnames of each remote host permitted to access your local server.

- If you are running as a stand-alone system, you must have your system's name in this file.
- If you are part of a network, the other system names must be included.

The syntax is as follows:

host
host
host

For example, if you are `hpaaaaa`, and regularly ran clients on `hpccccc`, and `hpddddd`, you would want the following lines.

```
hpaaaaa
hpccccc
hpddddd
```

Note that aliases work as well as hostnames, provided they are valid, that is, commonly known across the network.

X0.hosts and X0screens Relation

The default screen configuration file `X0screens` uses the default X11 remote host file `X0.hosts`.

Each custom `X*screens` file is associated with a special `X*.hosts` file. The number represented by the `*` causes the correct screen and host files to be used together. For example, `X3screens` takes an `X3.hosts` file. Both are referenced by the server when it is started with a `/usr/bin/X11/X :3` command.

If you use a special `X*screens` file, you need to set your `DISPLAY` variable appropriately. For the previous example, it would be set to `hostname:3.0`.

Using an /etc/hosts File

This file need not be present if your system is configured to query a nameserver.

The `/etc/hosts` file is an ASCII text file containing a list of all the host names and internet addresses known to your system, including your own system.

If your system is not connected to a network, use the loopback address (`127.0.0.1`) and the hostname `unknown`.

```
127.0.0.1  unknown
```

For a local system to access a remote host:

- The address and hostname of the remote host must be listed in the local system's `/etc/hosts` file.
- The user must have a valid login (username and password) and home directory on the remote host.

Software Configuration Files

The X Window System uses four configuration files:

`.Xdefaults`	Specified default appearance and behavior characteristics for clients. The contents of this file are covered in more detail in chapter 5.
`.x11start`	Specifies the clients that start when the X Window System starts. This file is covered in more detail in chapter 4.
`.mwmrc`	Specifies the menus, menu selections, button bindings, and keyboard bindings that control the OSF/Motif Window Manager (`mwm`). The contents of this file are discussed in chapters 5 and 7
`app-defaults/*`	Optional configuration for specific clients. The contents of this file are discussed in chapter 5.

If your home directory does not contain these files, the X Window System uses the system-wide versions of these files in `/usr/lib/X11`.

```
sys.x11start

system.mwmrc
```

If you want to customize your X environment, copy these files from the `/usr/lib/X11` to your home directory (noting the name change), and make your modifications. For example:

```
cp /usr/lib/X11/system.mwmrc $HOME/.mwmrc
cp /usr/lib/X11/sys.Xdefaults $HOME/.Xdefaults
```

The X server looks first in your home directory for these files. If they are not there, it uses the system-wide files.

Using Custom Screen Configurations

The default screen configuration is specified in the `X0screens` file in `/etc/X11`. It assumes:

- There is one display—display 0.
- There is one screen—screen 0.
- The screen uses Image mode (for older displays) or Combined mode (for newer displays)
- The screen is at the address node specified by `/dev/crt`.

If you use some configuration other than the default, you must edit the default screen file or add additional screen configuration files.

There should be a separate `X*screens` file for each display, where `*` is a number that matches the display number used when starting the X server. For example, `X0screens` is used for *hostname*`:0`, `X1screens` is used for *hostname*`:1`, and so on.

Creating a Custom 'X*screens' File

`X*screens` allows you to specify:

- device independent server options.
- screen device files.
- default visuals.
- monitor size.
- monitor power-saving level.
- device dependent screen options.

There are two ways to create a custom screen configuration for a display:

- You can modify `X0screens` so that it contains device information for all the screen configurations you may want to use. This is generally the preferred way. Only one configuration is used at a time; the others are commented out. To switch from one screen configuration to another, you uncomment some lines and comment others. For multiple displays, you would have a separate file for each display—for example, `X1screens` for display 1.
- You can have a separate `X*screens` file for each screen configuration on a particular display. Switching between them involves modifying the command that starts the X server.

X0screens Format

Entries in the `X0screens` file and any `X*screens` files are in the form:

```
[ServerOptions
   server_option
   ⋮
   server_option]
Screen device_name
   [DefaultVisual
      [Class visual_class]
```

```
        [Depth depth]
        [Layer layer]
        [Transparent]]
    [MonitorSize diagonal_length units]
    [ScreenOptions
        screen_options
        ⋮
        screen_options]
```

where:

`ServerOptions`	defines a block of options specific to a server
server_option	are X server options allowing the server to make the best use of your display.
`Screen`	defines a block of options specific to a screen.
device_name	is the name of the device file.
visual_class	describes the kind of grayscale or color to use as the visual.
depth	the number of planes (image mode only)
layer	the operating mode: image or overlay
`Transparent`	allow transparent windows (overlay mode only)
`MonitorSize`	use only if you have a non-standard monitor
`ScreenOptions`	defines a block of options specific to certain hardware.
screen_options	are options specific to different screens.

The `Screen` *device_name* line is the only required entry. If no other options are specified, the server will use the default options for that device.

The correct entries for your hardware are provided in the `/usr/lib/X11/Xserver/info/screens/hp` file (for HP-UX 10.0 and later systems).

The `XOscreens` file, located in the `/etc/X11/` directory provides detailed information about how to create your own `X*screens` file.

Operating Modes (Visual Layers)

Display hardware can have two kinds of display planes, image and overlay.

There are three possible server operating modes using these display planes. Different display hardware allows different modes, so all modes might not be available to you. In general, older devices can use all three modes, while newer ones use Combined.

On HP-UX 10.0 (and later) systems, for information about what modes your display hardware supports, read the `/usr/lib/X11/Xserver/info/screen/hp` file (HP-UX 10.0 and later systems).

The three screen modes are image, overlay, and combined:

Image Mode. The default screen mode using multiple image planes for a single screen. The number of planes ("depth") determines the variety of colors available to the screen.

An example of specifying image mode is:

```
Screen           /dev/crt
   ScreenOptions
      Layer      Image
```

Overlay Mode. An alternate screen mode using overlay planes for a single screen. You can see what is in the image planes only if you open a "transparent" window in the overlay planes and move the window over what you want to see. Typically, overlay planes are used in conjunction with image planes in combined mode.

An example of specifying overlay mode is:

```
Screen           /dev/crt
   ScreenOptions
      Layer      Overlay
```

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.

The `X*screens` entry for this modes is more complicated. A primary and secondary device are specified, each with their own mode.

An example, with `/dev/ocrt` as the primary device (running in overlay mode) and `/dev/crt` as the secondary device (running in image mode):

```
Screen                    /dev/ocrt
   ScreenOptions
      VRXSecondaryDevice  /dev/crt
```

Double Buffering

This feature applies to image planes only. Double buffering is not available on all displays. Refer to `/usr/lib/X11/Xserver/info/screen/hp` for information about your display.

Double buffering means that half of the color planes of your displays 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 also faster. However, double buffering ususally 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 buffer mode, the display will flash or flicker rapidly.

If you are using a recent display device, double-buffering required by applications will occur automatically.

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, 12, and 24. Valid depths for double buffered mode are 8, 16, and 24. The depth of overlay planes is determined by the `/dev` entry in `X*screens`.

For information about what depth your display hardware supports, read the `/usr/lib/X11/Xserver/info/screen/hp` file (HP-UX 10.0 and later systems only).

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.

Mouse Tracking with Multiple Screen Devices

If you use a multi-screen configuration, the mouse pointer can move from one screen to another. You can arrange the screens in a vertical, horizontal, or matrix orientation by adding the appropriate lines to the `X*pointerkeys` configuration file described in chapter 9. The sample `X*pointerkeys` file in `/usr/lib/X11` contains examples that show how to specify the orientation of multiple screens.

Note

The sample `X*pointerkeys` file is placed in `/etc/X11` at install time. If you subsequently update your system, the `X*pointerkeys` file in `/etc/X11` is *not* overwritten, and the sample file is placed in `/usr/newconfig`.

Moving the mouse pointer off one edge of a screen causes the pointer to move to another screen, depending on the screen orientation you have specified. In the 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 X11.

Other lines correspond to the screens that appear when the mouse is moved to the right or left side of the current screen. Moving off the right side goes to the next display listed, the left side the to previous display in the list. If you are on the first display listed and move right, you move to the last display listed. If you are on the last display and move left, you move to the first display.

Converting Old X*screens Files

The format of `X*screens` has changed at HPUX 10.0. Use the `convertscr` utility to convert an old `X*screens` file to the new format. Type:

```
/usr/bin/X11/convertscr -h
```

to learn how to use this utility.

Making a Device Driver File

Devices specified in screen configuration files must correspond to device files. If you don't have the appropriate device file, you must create it using the `mknod` command. For information on `mknod` see the system administration manual for your operating system.

Using Special Input Devices

Input devices are connected to Hewlett-Packard computers through several different hardware interfaces. Among the interfaces supported are the Hewlett-Packard Human Interface Link (HP-HIL) and the industry standard RS-232C (serial) and DIN interfaces. Some Hewlett-Packard computers do not support all of these interfaces.

How the Server Chooses the Default Keyboard and Pointer

The X server can access input devices through any of the above interfaces. Devices that use the HP-HIL interface and devices that use the DIN interface and that are compatible with the HP DIN keyboard and mouse can be used by simply plugging them into the computer. Devices that use the RS-232C interface require the installation of input device driver software before they can be used.

If no explicit input device configuration is done, the X server chooses the X keyboard device and X pointer device from the input devices that are connected to the computer (in most cases, the keyboard and a mouse). On computers that support both HP-HIL and DIN interfaces, the DIN input devices are used if both types of devices are connected.

HP-HIL input devices can plug into other HP-HIL devices, with up to seven input devices connected together. If there are no DIN input devices connected, and there are multile HP-HIL input devices, the following algorithm is used to choose an X keyboard and pointer device.

1. If no explicit specification is made through the `X*devices` file, the last mouse (the one farthest from the computer on the HP-HIL line) is used as the X pointer and the last keyboard is used as the X keyboard.
2. If no mouse is available, the last pointing device (such as a dial box, graphics tablet, or trackball) is used as the X pointer. If no keyboard is available, the last key device (such as a buttonbox or barcode reader) is used as the X keyboard.
3. If no pointing device is available, the last keyboard is used as the X pointer as well as the X keyboard.
4. If no pointer and keyboard are available, the X server won't run unless explicitly configured to run with no input devices.

X0devices File

The X server reads an input device file, `X0devices` in `/etc/X11`, to find out what input devices it should open and attach to the display.

Note

The sample `X0devices` file is loaded into `/etc/X11` unless one already exists. In that case, it is loaded into `/usr/newconfig/etc/X11`.

The default `X0devices` file 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.
- 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 `X*screens` file, you would not necessarily need an `X*devices` file.

A custom `X*devices` file is required only when you want to tell the X server about a custom input device configuration.

Explicitly Specifying Input Device Use

The X server can be explicitly configured to use a specific input device as the X pointer or X keyboard, or merge the data from an input device with that from the X pointer or keyboard. This configuration is done by adding information to the `X*devices` file. There is one syntax to use for HP-HIL devices, and another syntax for devices that require a device driver to be loaded by the X server (such as RS-232 devices).

HP-HIL devices can be specified in either of two ways:

- Device type and position.
- Device file name.

Explicitly specifying RS-232 Input Device Use

Some RS-232C input devices can be used with the X server. A device driver must exist for the desired serial input device, and it must reside in the `/usr/lib/X11/extensions` directory. Input device drivers are usually supplied by the input device vendor along with the input device. Sample input device drivers and documentation describing how to write an input device driver may be found in the `/usr/contrib/X11drivers/input` directory.

To use an RS-232 input device, you must modify the `X*devices` file to inform the X server which input device driver is to be loaded, the serial port to which it is connected, and how it is to be used. This is done by adding an entry to the `X*devices` file of the following form:

```
Begin_Device_Description
Name        device_driver_name
Path        device_file_path
Use         device_use
End_Device_Description
```

where:

device_driver_name Specifies the name of the input device driver shared library.

device_file_path Specifies the name of the device file for the serial port being used.

device_use Specifies the desired use of the input device, such as "keyboard", "pointer", "other", or "extension".

The following example specifies a Spatial System Spaceball ® connected to the serial port associated with device file `/dev/tty00` as the X pointer:

```
Begin_Device_Description
Name        spaceball.sl
Path        /dev/tty00
Use         pointer
End_Device_Description
```

More examples of input device specifications for RS-232 input devices are in the `/usr/newconfig/etc/X11/X0devices` file.

Specifying HP-HIL Input Device Use by Device Type and Position

The device can be specified using its device type and position by adding an entry to the `X*devices` file with the following form:

relative_position *device_type* *use* *#comments*

where:

relative_position Specifies the position of the device on the HP-HIL relative to the other devices on the HP-HIL, for example, "first", "second", and so on.

device_type Specifies the type of input device, such as "keyboard", "mouse", or "tablet".

use Is "keyboard", "mouse", or "other".

#comments Describes device. Comments are optional, but must start with a "`#`".

Valid positions, types, and uses are in "Selecting Values for 'X*devices' Files", along with examples.

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 the same type. For example if you have two keyboards, a graphics tablet, and a mouse connected, they are referred to as "first keyboard", "second keyboard", "first tablet", and "first mouse".

This syntax is useful for computers on which a single X server is running, and on which no other programs directly access input devices. With this syntax, 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 device ahead of the existing device.

This syntax should not be used if more than one X server will be run on the same computer, or if non-X programs will be directly accessing input devices. The X server interprets "first" to mean "first accessible", so you may not always get the first on the HP-HIL, just the first one not already in use.

Selecting Values for 'X*devices' Files

`X*devices` files use the following special names for positions, devices, and uses:

Values for 'X*devices' Files.

Positions	Device Type (Device Class)	Uses
first	keyboard (keyboard)	keyboard
second	mouse (pointer)	pointer
third	tablet (pointer)	other
fourth	buttonbox (keyboard)	
fifth	barcode (keyboard) **	
sixth	one_knob (pointer)	
seventh	nine_knob (pointer) *	
	quadrature (pointer)	
	touchscreen (pointer)	
	trackball (pointer) ***	
	null	

* 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 transmission 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 file name, 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.

Examples

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 a system with no input, include the following lines in the `X0devices` file:

```
first null   keyboard
first null   pointer
```

If you had a more complicated configuration, such as two graphics tablets, two keyboards, and a barcode reader, your `X*devices` file could look like this:

```
first    tablet     pointer     The pointer.
second   tablet     other       Merged with the pointer.
first    keyboard   other       Merged with the keyboard.
second   keyboard   keyboard    The keyboard.
first    barcode    other       Merged with the keyboard.
```

In this example, the first tablet acts as the pointer, the second keyboard acts as the keyboard, input from the second tablet is treated as if it came from the X pointer, and input from the first keyboard and the barcode reader is treated as if it came from the X keyboard.

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 would read as follows:

```
third keyboard    other
```

More examples can be found in the `X0devices` file in `/usr/newconfig/etc/X11`.

Specifying HP-HIL Input Device Use by Device File Name

The device can be specified using the name of the device to which it is attached. This can be done by adding an entry to the `X*devices` file with the form:

/path/ device_file *use* *#comments*

where:

path/device_file Specifies the name of the device file associated with the input device.

use is "keyboard", "pointer", or "other".

#comments Describes the device. Comments are optional, but must be preceded with a "**#**".

This syntax should be used if more than one X server will be running on the computer, or if non-X programs will be accessing the input devices. It refers to a specific position on the HP-HIL.

Redefining the HP-HIL Search Path

The `X*devices` file can be used to redefine the path searched for HP-HIL devices. By default, the path searched is `/dev/hil`. The device files are named by appending the numbers "1" through "7" to the path.

The path is redefined by adding an entry to the `X*devices` file with the following form:

path `hil_path` `#`*comment*

where:

path Specifies the path to be searched for the HP-HIL input devices.

#comments Describes the path. Comments are optional, but must be preceeded by a "`#`"/

The X server appends the numbers "1" through "7" to the specified path. For example, specifying:

```
/tmp/foo    hil_path
```

results in the device names `/tmp/foo1`, `/tmp/foo2`, and so on.

Customizing for Native Language Support (NLS)

This section covers:

- How X uses the `LANG` environment variable and other environment variables.
- Accessing language-dependent message catalogs and resource files.
- Remote execution in NLS systems.

Setting the LANG Environment Variable

The `LANG` environment variable must be set in order to use native language support. Setting `LANG` causes X to use the language-sensitive routines for character handling.

You can set `LANG` to any locale that your system supports.

To find out what locales are available to your system, type:

```
locale -a
```

To find your current language settings, type:

```
locale
```

To set the `LANG` variable:

```
LANG=language
export LANG
```

(This shows the `ksh` commands. If you are using another shell, use the appropriate command for setting environment variables in that shell).

This example sets the LANG variable to Spanish:

```
LANG=es_ES.roman8
export LANG
```

Other NLS Environment Variables

This section covers other NLS environment variables. It provides an overview only. For detailed information, refer to *X Toolkit Intrinsics Programming Manual*.

Message Catalogs—The NLSPATH Environment Variable

The NLSPATH environment variable determines the paths applications search for NLS message catalogs. X clients place NLS message catalogs in client-specific locations, allowing translated catalogs to be shared. For example, HP VUE's Vuepad places its catalog in the /usr/lib/nls/msg/$LANG directories.

It shouldn't be necessary to set NLSPATH unless the message catalogs are installed in non-standard locations.

The proper value of NLSPATH depends on whether message catalogs exist for the current value of LANG.

Setting the XUSERFILESEARCHPATH Environment Variable

The XUSERFILESEARCHPATH environment variable controls where X applications look for their app-defaults resource files. The default app-default location is :

```
/usr/lib/X11/%L/%T/%N%S:/usr/lib/X11/%l/%T/%N%S:/usr/lib/X11/%T/%N%S
```

If your app-defaults is in any other location, you need to set the XUSERFILESEARCHPATH, XAPPLRESDIR, or XFILESEARCHPATH variables described later in this section.

For example, to use Japanese app-defaults you would set XUSERFILESEARCHPATH to /usr/lib/X11/ja_JP.eucJP/app-defaults. Or, you could set XAPPLRESDIR to /usr/lib/X11/%L/app-defaults and LANG to "ja_JP.eucJP". If LANG is not set, %L defaults to null.

If you set XUSERFILESEARCHPATH in $HOME/.profile, the value applies to all X clients you run. Non-clients will not find their resource files unless you link or copy them into the directory specified by XUSERFILESEARCHPATH.

Setting the KBD_LANG Environment Variable

X allows you to override the physical keyboard attached to the HP-HIL.

Some applications use the environment variable KBD_LANG allowing you to change the keyboard mapping to that of another national language keyboard.

Note

Due to changing keyboard standards and the need for interoperability with non-Hewlett Packard system, this capability is being phased out. Do not depend on the `KBD_LANG` variable, since it may not be available in future releases.

This variable can be set after the server has started. The NLIO processes for Asian users start only when either the physical keyboard is Asian or `KBD_LANG` is set to an Asian language.

Language-Dependent Bitmaps—the XBMLANGPATH Variable

The `XBMLANGPATH` variable specifies the search path for language-dependent bitmaps. It lists the paths for bitmaps in this order:

1. User-specific bitmaps.
2. System bitmaps listed in the `XmGetPixmap(3x)` man page.
3. Append:

   ```
   /usr/lib/X11/bitmaps/%N/%B
   ```

   This ensures that you will get the non-localized bitmaps, where necessary.

Other Language-Dependent Resource Files

When `LANG` is set, `mwm` uses the following language-dependent default resource file:

```
/usr/lib/X11/%L/system.mwmrc
```

Native Language Fonts

For information about using non-English fonts, refer to "Using Native Language Input/Output" in chapter 6.

4

Using the X Window System

This chapter covers:

- Starting the X Window System.
- Stopping X clients.
- Exiting the X Window System.

The following chapters contain related information:

- Chapter 3 explains configuration files used by the X Window System.
- Chapter 7 explains the window manager (`mwm`) in more detail.

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.

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.

X will use the default `.x11start`, `.Xdefaults`, and `.mwmrc` files, unless told otherwise in the command line options.

Starting X at Login

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"

Starting X from the Command Line

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

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*]`--` [{*path*}*/server*] [*:display*] [*-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.

The default `.x11start` file starts the following clients:

- A terminal emulation client, such as `hpterm`.
- `mwm`.

Server options are preceded with a double hyphen (`--`). If the option following the double hyphen begins with a slash (/) or a path and 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. Refer to the Xserver man page for more information on server options.

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* `$@` *a blue background.*
`x11start -- /X2`	*Starts server X2 rather than the default server.*

Starting X on an HP-UX Multi-Display 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.

Seat 0 uses the `X0screens` and `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. To start X11 on seat 0 (display 0) of a multi-seat system, log in as usual and type:

`x11start` Return

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 `X1screens` and `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

Whether you start the X Window System from the command line or automatically from a login file, `x11start` always executes the same sequence of steps.

1. If necessary, it adds the system directory, `/usr/lib/X11`, to your `PATH` variable.
2. It looks in your home directory for a `.x11start` command file to read. If it doesn't find one, it reads `sys.x11start` in `/usr/lib/X11` instead.
3. It starts `xinit`, which starts the server and any clients specified in the `.x11start` command file.
4. It looks in your home directory for a `.Xdefaults` configuration file to read. If it doesn't find one, it reads `sys.Xdefaults` in `/usr/lib/X11` instead.
5. It reads the configuration file named by the `$ENVIRONMENT` variable, `.Xdefaults-`*hostname* if the variable doesn't exist.

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

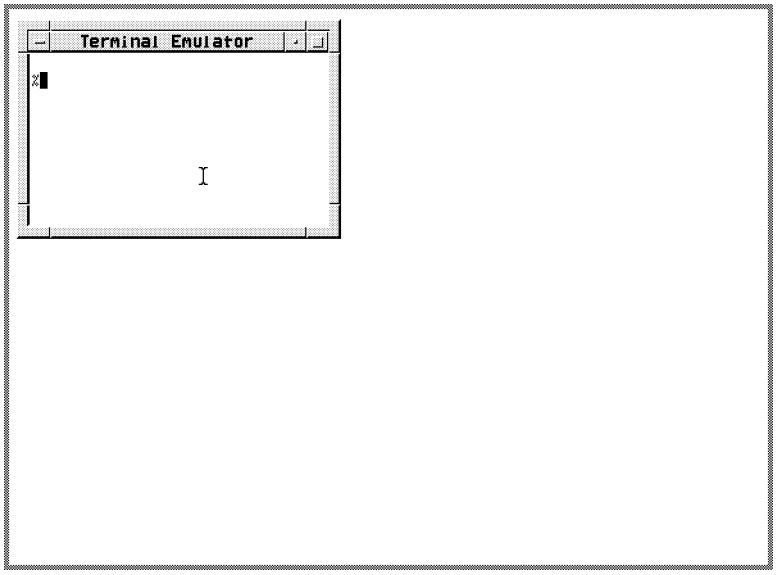
The Server Creates the Root Window

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.

In the center of the root window is an hourglass. 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 the pointer changes to an ×, and 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 OSF/Motif Window Manager (`mwm`), your window has a three-dimensional frame. This frame contains window manager controls.



The Default X Environment: 'mwm' and One Window

The window contains a command-line prompt and behaves exactly like the screen of a terminal. You can think of this window as "a terminal in a window." There are several terminal emulation clients: inclusing `hpterm`, `dtterm`, and `xterm`. The examples in this book use `hpterm`. Refer to the man page for your terminal emulator for specific details about it.

Move the mouse. The pointer moves on the screen. When the pointer is in the root window, it has an `X` 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 OSF/Motif Window Manager (operating in "explicit focus" mode), when you press and release button 1 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

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 the text following this table.
The root window displays, but no pointer appears.	Press CTRL Left Shift Reset all at the same time. (CTRL Left Shift Pause on PC-style keyboards.) This brings your original command-line prompt back. Read the text following this table.
The root window and pointer display, but no terminal window appears.	Press *and hold* button 3. If a menu appears, open a window. Otherwise, press CTRL Left Shift Reset (CTRL Left Shift Pause on PC-style keyboards.) Try restarting X, then read the text following this table if there's still a problem.
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) button 1, 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 that the correct directory is in your PATH statement. If you do not have an entry for the system directory, `/usr/lib/X11`, then `x11start` will add that entry to the path. You can be sure that the entry is always there by adding it to the path yourself. To check the PATH variable, type

  `env` Return

- Check that the `DISPLAY` environment variable is set correctly. If you do not already have an entry for either `local:0.0` or *host:0.0* (where *host* is the hostname of your system), X11 will add it for you when X11 starts. You can add the entry yourself. To check the `DISPLAY` environment variable, type:

  `env` Return

- Check that you have the correct permissions for the `.x11start` file in your home directory. Type:

  `ll .x11start` Return

  The resulting permission should be at least:

  ```
  -rwx------
  ```

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

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

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 before starting the X11 display server.

Before stopping the X Window System, you must first stop any X clients 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 X clients and any non-clients running in terminal emulator windows 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 three 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.

Closing the Window

You can also stop most applications by closing the window in which the application is running. To close a window:

1. Position the pointer on the window menu button.
2. Press *and hold* button 1.
3. Drag the pointer to Close.
4. Release button 1.

Stopping the X 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. (If you have a PC-style keyboard, press Shift Control Pause instead.)

The sequence of keys that stops the display server can be customized in the `X*pointerkeys` file. Refer to chapter 9 or the `XOpointerkeys` file in `/usr/lib/X11`.

5

Application Resources

Resources are data used by applications to set their appearance and behavior.

This chapter covers:

- The various ways to change resource settings.
- The scope of resources— how specifically or generally a resource is applied.
- The syntax for specifying color and geometry resources.

How Applications Obtain Attributes

An application can get attributes from several different places:

- Resources directly loaded into an application's resource database:
  - Command-line options.
  - `.Xdefaults` file.
  - Resources loaded into the RESOURCE_MANAGER property.
  - Application resource files (for example, `app-defaults` files or `.rc` files).
- Other sources:
  - Defaults built into the client.
  - Environment variables.
  - Inter-client communications.

The following list shows how applications obtain resources. A resource at the top of the list overrides the same resource found further down the list. For instance, a resource in `.Xdefaults` overrides the same resource in the `app-defaults` directory.

- Command-line 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 resource file.
- A host environment:
  - If an `XENVIRONMENT` variable exists, it may contain the name of a file that specifies application attributes.

  - A `$HOME/.Xdefaults-`*host* file may contain resources to be used for a specific remote host. It is read only if no `XENVIRONMENT` variable exists.
- Personal resources:
  - Loaded into the RESOURCE_MANAGER property.
  - `.Xdefaults` (or `sys.Xdefaults`) file.
- User-specific files for particular classes of applications:
  - If an `XUSERFILESEARCHPATH` variable exists, it may specify a directory of files containing application class defaults for the specific user.
  - If `XUSERFILESEARCHPATH` variable does not exist, and if an `XAPPLRESDIR` variable exists, it may specify a directory of files containing user-specific application class defaults.
  - `$HOME/`*app-class* files may contain application resources. These files are read only if `XUSERFILESEARCHPATH and XAPPLRESDIR` do not exist.

  For information about these variables, refer to *Programming with the Xt Intrinsics*.
- Application-specific configuration files in the `app-defaults` subdirectory of `/usr/lib/X11`. Each file specifies attributes for a particular class of application. An `app-defaults` file is the equivalent of a *Class*resource* statement in a resource file. (The environment variable `XFILESEARCHPATH` may define a language-dependent location of `app-defaults`.)
- Internal defaults built into the application.

Ways to Change Resources

There are several ways to change a resource. The way you choose depends on:

- The nature of the resource.
- When you want the change to take effect—immediately or at the beginning of the next session.

Resources can be changed by:

- Loading the new resources into the server's RESOURCE_MANAGER property using the X client `xrdb`.
- Hand editing a resource file, such as `.Xdefaults`.
- Using command-line options.

Setting Resources with .Xdefaults

The `.Xdefaults` file contains default resources you want to apply each time a client is started.

If you do nothing, the system uses the the `sys.Xdefaults` file in `/usr/lib/X11`. If you want your own defaults to be used instead, copy this file into `.Xdefaults` in your home directory and make modifications there. For example:

```
cp /usr/lib/X11/sys.Xdefaults $HOME/.Xdefaults
```

The syntax for describing resources is explained later in this chapter.

Changing the RESOURCE_MANAGER Property with 'xrdb'

The RESOURCE_MANAGER property is a property on the root window that is treated the same way as a resource file by the resource manager.

During a session, the RESOURCE_MANAGER property may be modified by the the `xrdb` client.

You can use `xrdb` to load resources into the server's RESOURCE_MANAGER property.

The syntax for `xrdb` is:

`xrdb` *options* [*filename*]

Where *options* are:

`-help`	Displays a list of options for `xrdb`.
`-display` *host: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` *path/filename*	Specifies that `xrdb` should load the file named on the command line into the RESOURCE_MANAGER property, overwriting the current resources listed there. This is the default action.
`-merge` *path/filename*	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` *path/filename*	Places the contents of the RESOURCE_MANAGER property into the

	named file, overwriting resources specified there.
`-backup` *string*	Specifies a suffix to be appended to the filename used in the `-edit` option to create a backup file.
`-cpp` *path/filename*	Specifies the path and filename of the C preprocessor to use when loading a configuration file containing `#ifdef` or `#include` ents. `xrdb` works with CPP and other preprocessors as long as they accept the `-D`, `-U`, and `-I` options.
`-nocpp` *path/filename*	Specifies that `xrdb` should not use a preprocessor before loading the configuration file (the file contains no statements that need preprocessing).
`-symbols`	Displays the symbols currently defined for the preprocessor.
`-D`*name*[=*value*]	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.

To add resources interactively:

1. Execute:

   ```
   xrdb -merge -nocpp
   ```

   in a local terminal emulation window.

2. Type in the resource specifications. Each resource must be on a separate line.

3. When you've typed all the resources, press CTRL d to merge the resources and restore the shell prompt.

To add resources by typing the resources into a file that is then merged into the database:

1. Create a file containing the resources you want to add.

2. Execute:

   `xrdb -merge -nocpp` *filename*

Syntax of Resource Specifications

Resource files are text files. They must obey the following syntax rules:

- Each resource specification must be on a separate line. If the last character on a line is a backslash (\), the new-line following the backslash is ignored and the resource specification is assumed to continue on the next line.
- To add comments to resource files:
  - Use the exclamation (!) character. Anything to the right of the ! is interpreted as a comment. This is the preferred way of commenting all or portions of lines.
  - You can place a pound (#) character in column 1. This makes the entire line a comment. Keep in mind that you must use the `xrdb` option `-nocpp` when loading a commented resource to avoid it being interpreted as a preprocessor directive.
- The resource name is separated from the value by a colon (:) and optional spaces or tabs.

The general syntax for specifying a resource for a client is:

$$\left[\left\{ \begin{matrix} client\_name \\ client\_class \end{matrix} \right\} \right] * \ resource: \ value$$

For example:

```
hpterm*background: skyblue
```

sets the background color of the `hpterm` window to skyblue.

Trailing blanks in a resource value are parsed and therefore can cause errors to occur. For example, if you inadvertently included a blank after "skyblue" in the example above, several warning messages appear when the program using the resource definition is run:

```
Warning: Color name "skyblue " is not defined
Warning: Cannot parse default background color spec
```

Certain clients allow you to set resources for particular parts of the client. For example,

```
hpterm*scrollBar*background: mediumblue
```

sets the scrollbar on `hpterm` windows to mediumblue.

Scope of Resource

You can specify how generally or specifically a resource is applied. For example, you can specify that all clients have a background color of black (very general). At the other extreme, you can say that you want the softkeys of one particular `hpterm` window to be red.

Scope of customization is determined by:

- Using names or classes of clients.
- Using names or classes of resources.
- Specifying particular areas of clients (for example, softkeys and scrollbars).
- Using wildcards in the resource string.

Names and Classes of Clients

Every client has both a name and a class. The name defines the specific client, while the class categorizes the client. Thus, the class is more general than the name.

Frequently, the two identifiers are very similar, and often differ only in capitalization. For example, the client named `xclock` belongs to class `Xclock`.

Resources specified by client name take precedence over resources specified by client class.

Naming a Client

You can assign a name to a particular instance of a client. This allows you to allocate resources to that client by class, by client, *and* by name.

For example, the following command line starts an instance of `hpterm` named `localTerminal`.

```
hpterm -name localTerminal
```

If the following resource exists in the resource database:

```
HPterm.name:              localTerminal
localTerminal*background  white
```

then the `localTerminal` window will be white, overriding the colors used by the current palette.

Names and Classes of Resources

Like clients, resources have both a name and a class.

An individual resource begins with a lowercase letter. For example, `foreground` refers to the foreground resource. A class resource, however, begins with an upper-case letter. For example, `Foreground` refers to the entire class of foreground resources.

Thus, if no other specifications overruled, the line `*foreground: blue` in your resource 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, frames, icons, and mattes.

Name/Class Precedence

Specific resource specifications always have precedence over general specifications. For example, suppose a resource file contains:

```
*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, while the other resources of class Foreground (pointerColor, cursorColor, softkey foreground and bottomShadowColor, and scrollbar foreground and bottomShadowColor) remain DarkSlateGray for `hpterm` clients.

Similarly, if a resource file contains:

```
hpterm.name:                    local
HPterm*softkey*background:      wheat
HPterm*background:              pink
local*background:               white
```

then all softkey backgrounds will be wheat. For the rest of the `hpterm` window, the backgrounds will vary. Windows named `local` will be white, other windows will be pink.

Wildcards and Exact Paths

The * character in a resource string is a wildcard that provides resource generality. For example, the following list of resources shows increasing specificity.

```
*foreground:                    white
hpterm*foreground:              yellow
hpterm*softkey*foreground:      red
```

The resource `*foreground` refers to *all* foregrounds. The more specific resources override it. All the `hpterm` foregrounds will be yellow except for the foreground of the softkeys.

Color Resources

You can specify color resources in either of two ways:

- By color name.

  The `rgb.txt` file in `/usr/lib/X11` lists all the named colors. Refer to "Creating a Custom Color Database with 'rgb'" in Chapter 8 for information about how to add colors to this file.

- By a number specifying the amount of red, green, and blue the color contains.

  The rgb numbers have the syntax:

  `#`*RedGreenBlue*

  where *Red*, *Green*, and *Blue* are hexadecimal numbers containing 1, 2, 3, or 4 digits for each primary color indicating the amount of that color used. There must be 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, white can be specified by any of these rgb values: `#fff`, `#ffffff`, `#fffffffff`, or `#fffffffffff`. Red can be specified by `#f00`, `#ff0000`, `#fff000000`, or `#ffff00000000`.

The following line specifies the background color of `hpterm` icons by color name:

```
Mwm*hpterm*iconImageBackground:   DarkSlateGrey
```

The same color could be specified by rgb value:

```
Mwm*hpterm*iconImageBackground:   #2f2f4f4f4f4f
```

Refer to the man page for a specific client to see if there are special elements for that client that can be colored. For example, `xclock` allows you to color the hands and tic marks in addition to the background, foreground, and window frame colors.

Geometry Resources

The geometry of a window is its size and location. The syntax for geometry resources is:

$$\left\{ \begin{array}{l} Width \times Height \\ \pm column \pm row \\ Width \times Height \pm column \pm row \end{array} \right\}$$

Use a lower-case `x` for the times sign.

Width — The width in characters (for terminal windows) or pixels (for other clients). For widths in characters, the window size depends 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 depends on the font.

column The column location of the window in pixels.

- Plus (+) values: The location of the left side of the window relative to the left side of the workspace.
- Minus (−) values: The location of the right side of the window relative to the right side of the workspace.

row The row location of the window given in pixels:

- Plus (+) values: The location of the top of the window relative to the top of the workspace.
- Minus (−) values: The location of the bottom of the window relative to the bottom of the workspace.

Example Locations for an 80×24 Terminal Window.

To position a window here ...	Use this location ...
The upper left corner of the workspace.	+1+1
The lower left corner of the workspace.	+1−1
The upper right corner of the workspace.	−1+1
The lower right corner of the workspace.	−1−1

For example, the following line specifies that all `hpterm` windows be created 80 characters wide and 24 characters high, and that they are initially placed in the upper right corner of the display.

```
hpterm*geometry:   80x24-1+1
```

Font Resources

There are four general font resources that are commonly used.

General Font Resources

Resource	Description
`Font`	General user font
`FontList`	Displayed in system areas of clients created using the OSF/Motif toolkit.
`XmText*FontList` `XmTextField*FontList`	Displayed in text entry boxes of clients created using the OSF/Motif toolkit.

Use the following syntax to specify font resources:

{ *client_class client_name* }**fontresource*: *fontname*

where:

client_class The class of the client for which you specify the font.

client_name The name of the client for which you specify the font.

fontresource The name of the font resource.

fontname The name, alias, or xlfd name of the font. Refer to chapter 8 for information about how to specify font names.

For example,

`hpterm*Font:` *fontname*

Font resources and names are covered in more detail in chapter 6.

6

Using Fonts

This chapter covers:

- Displaying samples of bitmapped fonts and scalable typefaces.
- Setting font resources.
- Using the X11R5 font server.
- Understanding and using the XLFD (X Logical Font Description) font name for bitmapped fonts and scalable typefaces.
- Administering bitmapped fonts and scalable typefaces.

Chapters containing related information are:

- Chapter 5 covers where and when to set resources.
- Chapter 7 covers running clients from the command line.

A **font** is a type style in which text characters are printed. The X Window System includes a variety of fonts.

Bitmapped fonts are made from a matrix of dots. The font is completely contained in one file. Many files are needed to have a complete range of sizes, slants, and weights. Bitmapped font files can be read by the X server or the font server.

Scalable typefaces are each defined by a file containing a mathematical outline used by the system to create a bitmapped font for a particular size, slant, or weight. Scalable typefaces are readable by the font server. An X server wishing to use them must obtain them from a font server. See the sections entitled "Scalable Typeface Administration" and "Scalable Typefaces File Structure" in this chapter for more information.

Hewlett-Packard's X11R5 release of the font server supports two scalable font technologies: Agfa's Intellifont and Adobe's Type 1. Scalable outlines bundled with your operating system include Agfa's "CG Times," "Univers," and "Courier," and Adobe's "Utopia" and "Courier."

In addition to the scalable font technology available with the X11R5 font server, both the X server and the font server are now capable of rescaling bitmapped fonts to any size. This is not a recommended method of creating new fonts from existing ones — the results are often unsightly or even unreadable — but it is occasionally useful. The discussion of scaled fonts below also applies to scaled bitmaps, except where indicated.

The Intellifont Scalable Typeface Library available from Agfa includes hundreds of different designs. Please call Agfa directly at 1-800-424-TYPE (8973) for more information about Intellifont typeface products.

Customizing the Font Search Path with 'xset'

The X server must know where to find the fonts you want to use. The *font path* is a list of font sources accessible to the X Window System. A font source can be either a directory containing bitmapped fonts, or a font server accepting connections at some TCP address.

The `xset` command allows you to tell the X server which font sources to use. You specify directories containing bitmapped fonts by the complete path name. You specify font servers by the string "tcp/<hostname>:portnumber." To examine the font path, type:

```
xset q (Return)
```

To add or remove sources from the path:

```
xset options
```

where the options are:

`-fp` *source[,source...]* `fp-` *source[,source...]*	Remove the directories from the head (-fp) or tail (fp-) of the font path.
`+fp` *source[,source...]* `fp+` *source[,source...]*	Adds the sources to the head (+fp) or tail (fp+) of the font path.
`fp=` *source[,source...]*	Specifies the complete font path. The "`=`" is optional.
`fp default`	Resets the default font path.
`fp rehash`	Causes the server to reread the font databases for all directories (but not font servers) in the font path. This should be done after making any changes to directories that are in the font path, especially if you run `mkfontdir`, or if you change

`fonts.alias` in any of these directories.

Here are some examples that show you various ways to use `xset`.

```
xset fp tcp/:7000
```

Tells the X server to get all fonts from a font server running at TCP address 7000 on the local host.

```
xset fp+ tcp/fontmaster:7000
```

Tells the X server to add a font server running on host `fontmaster` at TCP address 7000, to the end of the font path.

```
xset +fp /usr/lib/X11/fonts/misc
```

Tells the X server to add the directory `/usr/lib/X11/fonts/misc`, which contains bitmapped fonts, to the beginning of the font path.

More information about `xset` is presented in chapter 8.

Listing Available Fonts with 'xlsfonts'

The `xlsfonts` client lists fonts available to you. It uses the `fonts.dir` and any `fonts.alias` files in the font search path to find the fonts. The XLFD name or the alias name is listed. Refer to `***<xref XLFD>: undefined***` for information about the XLFD name, and to "The fonts.alias File" for information about alias names.

The `xlsfonts` client has the following syntax:

`xlsfonts` [*-options*]

where *options* are:

`-display` *host:display*	The X server whose fonts you wish to list. The default is the requesting display.
`-l`	Generate a medium listing.
`-ll`	Generate a long listing.
`-lll`	Generate a very long listing, showing individual character metrics.
`-m`	Long listings should show minimum and maximum bounds of each font.
`-C`	Multiple column listings. Same as `-n 0`.
`-1`	Single column listings. Same as `-n 1`.
`-w` *width*	Width in characters of each column. Default is 79.
`-n` *columns*	Number of columns for listings.
`-u`	Output is unsorted.

`-o`	Use `OpenFont` and `QueryFont` rather than `ListFonts`.
`-fn` *pattern*	`xlsfonts` will find all fonts that match this pattern. Wild cards may be used. If this option is not included `xlsfonts` lists all available fonts.

An example listing looks like this:

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-hp-roman8
-adobe-courier-bold-o-normal--12-120-75-75-m-70-hp-roman8
⋮
courb10
courb12
⋮
```

The first two lines show the fonts' XLFD names, and the second two lines show the file name aliases for those fonts.

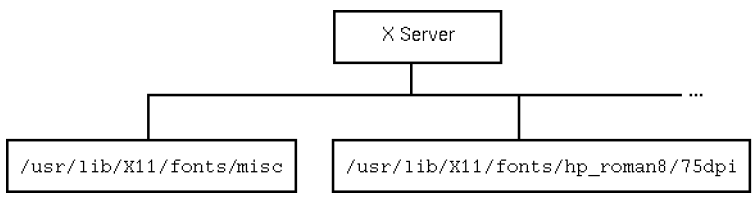
If you have many fonts on your system, `xlsfonts` can produce a long list. If you want to check for a specific font, use the pattern matching capability of `xlsfonts`. Use wild cards to replace the parts you are not trying to match. For instance, to see what scalable typefaces you have, type:

```
xlsfonts -fn "*-0-0-0-0-*" Return
```

Using the X11R5 Font Server

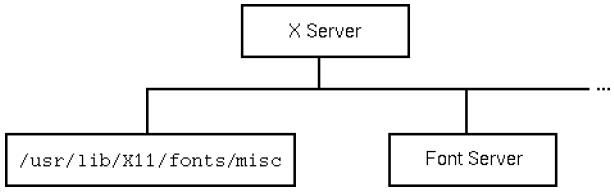
The Networked font server provides font services to one or more X display servers in a networked environment. It allows a font administrator to distribute fonts to all X servers from a central administration point. The font server also provides increased font capabilities over those built into the HP X display server — unlike the X display server, the font server understands scalable fonts as well as bitmapped fonts.

In an environment without a font server, the X server is able to load bitmapped fonts from directories in its font path:



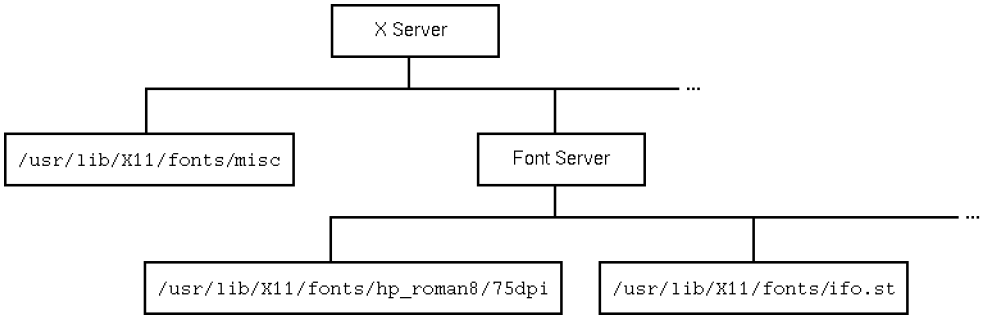
An Environment Without a Font Server

In an environment with a font server, the X server can also load fonts from a font server:



An Environment With a Font Server

The font server itself obtains fonts from directories or from other font servers. Unlike the X server, the font server can read scalable fonts as well as bitmapped fonts:



Font Server Reads Scalable and Bitmapped Fonts

The connection between the X server and a font server is over a TCP network connection: the font server can be running on the same machine as is the X server, or on a remote machine that is acting as a font source for multiple X servers.

A font server is shipped with Hewlett-Packard's X11R5 distribution. If the system has not been configured to start the font server automatically (refer to "Starting the Font Server at Boot Time"), then the font server can be started automatically with the command:

```
/usr/bin/X11/fs -daemon
```

This starts a font server in its default configuration at its default TCP port of 7000. Any X server started on the same system after the font server is started will automatically gain access to the fonts provided by that font server, including the licensed scalable Intellifont and Type 1 fonts in the `/usr/lib/X11/fonts/ifo.st` and `/usr/lib/X11/fonts/type1.st` directories.

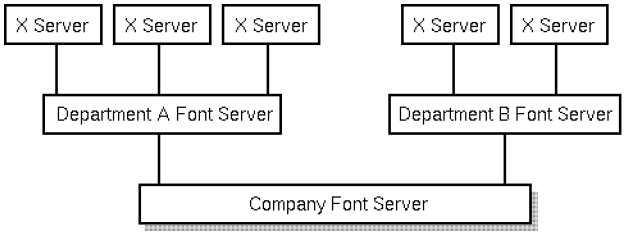
Managing the Font Server's Configuration

By default, a font server accepts connections from font clients (such as X servers) at TCP address 7000, and configures itself according to information in the file `/etc/X11/fs/config`. Both of these defaults can be overridden with command-line options. See the fs(1) man page for more information.

Like the X server, a font server has a font path, a list of sources from which it can get fonts. There are three major differences between the font server's font path and the X server's font path:

- The font server font path is specified in the configuration file, and cannot be changed by a protocol request from a font client. For example, an X server can add or remove font servers to or from its font path, but it cannot tell a font server to change its own font path.
- The font server font path can be changed by modifying the configuration file and sending a signal SIGUSR1 to the font server.
- The font server font path can include directories containing scalable fonts, such as `/usr/lib/X11/fonts/ifo.st` and `/usr/lib/X11/fonts/type1.st`.

Like an X server, a font server can have font servers in its path. That is, font servers can be "chained." Font server chaining can be used to implement powerful and flexible networks of font sources. For example, figure 6-4 shows a company-wide font server with department-specific font servers.



Chaining Font Servers

All users in Department A and Department B add their respective department's font server to their font path. The font administrator on each machine serving fonts then decides how to configure that font server for use by that department. In this case, he adds the company font server to the font path.

You can modify this path, as well as other configuration parameters, by editing the configuration file. See the fs(1) man page for information about all of the options in the configuration file.

Starting the Font Server at Boot Time

The behavior of the font server at system boot time is determined by system configuration files. Be default, the font server does not automatically start at boot time. You can change this behavior with the `mk_fnt_srvr` command.

If boot-time startup of the font server is enabled, the resulting font server will run at the default font server TCP address of 7000, and use the default configuration file `/usr/lib/X11/fs/config`.

Refer to the `mk_fnt_srvr(1M)` man page for more details.

The standard X interface provides a detailed description of the font by means of the *X logical font description* (XLFD) name. The XLFD name is a string of characters that describes properties of the font you want.

In X11R5, the XLFD standard supports both bitmapped and scalable fonts. In addition, HP has extended the standard to provide more capabilities with scalable fonts — that is, the ability to generate more variations on scalable fonts. These extensions are described in the following sections.

The form of the XLFD is 15 fields separated by dashes. These fields are explained later in this section.

"*FontNameRegistry-Foundry-FamilyName-WeightName-Slant-SetwidthName-AddStyleName-PixelSize-PointSize-ResolutionX-ResolutionY-Spacing-AverageWidth-CharSetRegistry-CharSetCoding*"

For example,

```
-adobe-courier-bold-o-normal--10-100-75-75-m-60-hp-roman8
```

specifies a courier, bold, oblique bitmapped font created by Adobe. The font is 10 pixels tall, 100 tenths of a point tall on a 75dpi×75dpi display. Characters are monospaced, and are an average of 60 tenths of a pixel wide. Fonts codes are based on the HP Roman8 encoding.

What is actually in the XLFD name differs depending on where in the font-request process the string is being used:

reference XLFD
: This is the XLFD name shown by `fonts.dir` and the `xlsfonts` client.

  Scalable typefaces have the PixelSize and PointSize fields set to zero.

request XLFD
: This is the XLFD name you use to request a font. It is also the XLFD name you use in a `fonts.alias` file.

  Any field in the list can be replaced by the "`*`" wild card. Any character in the list can be replaced by the "`?`" wild card.

resolved XLFD
: This is the XLFD name that the server returns when it has filled your font request.

All the fields are filled in with the correct values for that font. However, they may not be the same values as in the request XLFD.

XLFD Syntax

This section explains the meaning of the fields in the XLFD name. Examples of the use of these fields are in a later section.

The XLFD name is long, so you can assign a shorter nickname, or alias, for the font, which you then use in place of the long string. Aliases are discussed in "The fonts.alias File" later in this chapter.

You may use either upper-case or lower-case letters when you specify a characteristic. Reference XLFD names are all lower-case.

The text "*[extensions]*" means that there are optional extensions to the standard XLFD fields that are used to generate additional font variations. Notice that the underscore (_) character is used in some extensions to avoid confusion with the dash (-).

Note

These HP extensions may be superceded in later releases the X Windows System, as the industry-standard XLFD is expanded to include the functions that are now available only through these extensions.

FontNameRegistry

The authority that registered the font. Usually left blank. If there is a value in the field, it is of the form +*version*, where *version* is the version of some future XLFD specification.

Foundry

The name of the digital type foundry that digitized the font data.

FamilyName

The trademarked commercial name of the font. If the FamilyName contains spaces, do one of the following for a request XLFD name:

- Enclose the entire XLFD name in double quotes ("). For example, this `fonts.alias` file line.

```
italic "-agfa-cg century schoolbook italic-normal-i-*-*--240---p-150-*-roman8"
```

- Use wild cards for part of the field.

```
italic -agfa-*schoolbook*italic-normal-i-*-*--240---p-150-*-roman8
```

WeightName*[extensions]*

The relative weight of the font, such as bold.

For scalable typefaces, the user may specify that the font be darker (bolder) or lighter than the normal for that font. The syntax for this optional extension is:

$$\left[\pm horiz\_value \left[\pm vert\_value\right]\right]$$

horiz_value, *vert_value* — The increase (+) or decrease (-) in boldness. A value of 4000 for a normal font simulated the bold version of that font.

If only one delta and value are specified, they apply to both directions. Emboldening and lightening are currently supported only for Intellifont scalable typefaces.

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABCDEFGHIJKLMNOPQRSTUVWXYZ

The Same Font at Increasing Weights

Slant*[extensions]*

A code indicating the direction of the slant for the font.

`r`	Roman (no slant)
`i`	Italic (slant left)
`o`	Oblique (slant left)
`ri`	Reverse italic (slant right)
`ro`	Reverse oblique (slant right)

For scalable typefaces, the user can request additional slanting from the normal. The syntax for this optional extension is:

$\pm value$

$\pm value$ — The angle in 1/64 degree ranging from 0° to 75° (0-4800). (0.5° = 32, 1° = 64, etc) Values outside of that range will be truncated to ±75°. Use + for counterclockwise angles, - for clockwise angles.

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

The Same Font with Different Slants

SetwidthName

The width-per-unit of the font, such as compressed or expanded.

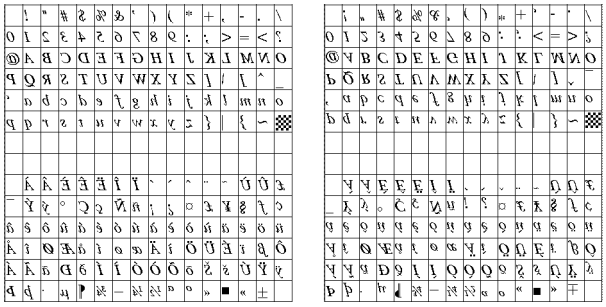
AddStyleName*[extensions]*

A description of anything else needed to uniquely identify the font, such as serif or cursive.

For scalable typefaces, users can specify that the font be mirrored or rotated. The syntax for the optional extension is:

$$\begin{bmatrix} \texttt{+Mx} \\ \texttt{+My} \end{bmatrix} [\pm \textit{angle}]$$

`+Mx`, `+My`	`Mx` mirrors the font horizontally, and `My` mirrors the font vertically.
angle	± the amount of rotation from normal in 1/64th degree increments. Use `+` for counterclockwise angles; use `_` for clockwise angles.



Font Mirrored Horizontally and Vertically

Don't confuse "slant" with "rotation". A character that has been slanted has its base in the normal position and the top pushed to one side. A character that has been rotated has been moved around some central pivot point.

PixelSize *[Extensions}*

An integer describing the height of an EM square in pixels.

For scalable typefaces, you can increase or decrease the horizontal size to make a font wider or narrower than normal for that font. The syntax for this optional extension is

$$[+\textit{pixelwidth}]$$

pixelwidth	The horizontal size in pixels. If this field is not specified, it is assumed to be the same as PixelSize.

For example, `20+10` requests a font 20 pixels high and 10 pixels wide (or, more accurately, a 20-pixel font whose width is half its normal width).

The PixelSize and PointSize fields are related through the ResolutionY field in the XLFD name (see below). You should specify

a font by using either PixelSize or PointSize, but not both. An error occurs if you specify both and they conflict.

PointSize*[extensions]*

An integer giving the EM square size in decipoints. For example 140 is 14 points.

For scalable typefaces, you can expand the horizontal size (set size) to make a font wider or narrower than normal for that font. The syntax for this optional extension is:

[*+setsize*]

+setsize The horizontal size in decipoints. If this field is not specified, it is assumed to be the same as PointSize.

For example, "140+240" requests a font 14 points high, and 24 points wide (or, more accurately, a 14-point font whose width is that of a 24-point font).

If neither PixelSize or PointSize are specified, the assumption is 12-point. If both are specified and they conflict, an error is returned. Use *either* PixelSize or PointSize, but not both.

The Same Font in Different Sizes

ResolutionX, ResolutionY

The horizontal (X) and vertical (Y) resolution of the device that the font was designed for, measured in pixels-per-inch. If the resolution is not specified in a request XLFD name, the X server defaults to the resolution of the display for which the font is requested.

Spacing

A code indicating the spacing between units in the font.

`M`	Monospaced (fixed pitch)
`P`	Proportional spaced (variable pitch)
`C`	Character cell. The glyphs of the font can be thought of as "boxes" of the same width and height that are stacked side by side or top to bottom.

AverageWidth

An integer string giving the average, unweighted width of all the glyphs in the font, measured in 1/10th device-dependent pixels.

CharSetRegistry

The registration authority that registered the specified CharSetEncoding. The XLFD conventions expect organizations that control characters to register with the X Consortium and be given a unique name to use in this field.

CharSetEncoding*[extensions]*

The character set from which the characters in the font are drawn.

For scalable typefaces, this field can be used to specify subsets of any of the character sets. This is a performance hint that the X or font server uses to determine which characters need to be realized. The syntax for this optional extension is:

=*value,value...*

value A character or range of characters to be included in the font, specified as decimal or hex number[s]. A range is two numbers separated by a colon (:). For example,

```
=65,0x45,80:85
```

specifies the characters "A," "E," and "P" through "U."

If an application requests a character not in the subset, then:

- If the font's usual default character (typically space) is in the subset, that character will be substituted.
- Otherwise, the result is font-dependent.

Using the XLFD Font Name

You use the XLFD name or alias whenever you need to specify a font. Some locations are:

- Application default or resource files, for example:

  `hpterm*Font:` *fontname*

- Command line to start clients or applications, for example:

  `xclock -digital -fn` *fontname*

- The `fonts.dir` file. Refer to "The fonts.dir File".
- The `fonts.alias` file. Refer to "The fonts.alias File".

The fonts.dir File

In directories containing scalable and/or bitmapped fonts, the X and font servers associate each font file name with the XLFD font name by means of the `fonts.dir` file. This file is created by the installation process, or by executing the `mkfontdir` (for directories that contain only bitmapped fonts), or `stmkdirs` (for directories that contain bitmapped and/or scalable fonts) utility. The `stmkdirs` utility is described in more detail in the section on font administration.

You should run `mkfontdir` or `stmkdirs` after you add or delete fonts from a directory, so that the change is reflected in `fonts.dir`. You can view the list of fonts in a font directory, and their XLFD names, by examining the `fonts.dir` file in that directory.

Scalable typefaces listed in `fonts.dir` have some values set to zero.

A `fonts.dir` file looks similar to this:

```
7
helv08.pcf.Z -adobe-helvetica-medium-o-normal--8-80-75-75-P-47-hp-roman8
helvB08.pcf.Z -adobe-helvetica-bold-o-normal--8-80-75-75-P-48-hp-roman8
helvR08.pcf.Z -adobe-helvetica-medium-r-normal--8-80-75-75-P-46-hp-roman8
ant_oliv.ifo -agfa-antique olive bold-bold-r-normal-91118-0-0-0-0-p-0-hp-td00000000
ant_oliv.ifo -agfa-antique olive compact-normal-r-compact-91120-0-0-0-0-p-0-hp-td00000000
ant_oliv.ifo -agfa-antique olive italic-normal-i-normal-91846-0-0-0-0-p-0-hp-td00000000
ant_oliv.ifo -agfa-antique olive-normal-r-normal-91119-0-0-0-0-p-0-hp-td00000000
```

In this example:

- The first line lists how many bitmapped fonts and scalable typefaces are described by the file, in this case 7.
- The rest of the lines give the file name and XLFD name that describes the file.
  - The 3 lines starting with `helv ...` are 3 different bitmapped fonts. They are different versions of the "Helvetica" style made by Adobe. They are all 8-points in size, but differ in the slant and boldness.
  - The last 4 lines are 4 scalable typefaces. Several fields in the reference XLFD name are set to zero. In the request XLFD

name you use to request one of these fonts, you supply values for either the PointSize or the PixelSize.

The X server tries to match your request with the bitmapped fonts and scalable typefaces listed in the `fonts.dir` file as follows: The server looks in the directories in your font path in the order shown by `xset q`.

- Bitmap fonts.

  The server uses the first font that it finds that meets all the criteria you specified in the XLFD name. If you specified everything, it will try to find the exact match. If you used the wild cards (`*` or `?`), it will use the first font that matches the parts you did specify.

- Scalable typefaces.

  The PixelSize, PointSize, ResolutionX, and ResolutionY fields in the reference XLFD name are zero. Your request XLFD name should specify either the PixelSize or PointSize (but not both). The server returns a font made from the outline with the specifications you requested.

If none of the above results in a font being returned, the X server returns an error message.

The fonts.alias File

A font can be referred to by an alias. The alias is shorter and easier to remember (and type) than the complete font description. Aliases are found in the `fonts.alias` file. The `fonts.alias` file need not be in each font directory, but the directory containing it must be in the font path.

A simple `fonts.alias` file is created as part of installing the font. The `fonts.alias` file is in this format:

```
"FILE_NAME_ALIASES"
alias-name  xlfd-name
```

where:

alias-name	is the nickname for the font.
xlfd-name	is the XLFD name that specifies the font. If the family name contains spaces, enclose the whole XLFD name in quotation marks(").

The `fonts.alias` file provides for two types of alias names:

- The font's file name.

  If the string `"FILE_NAMES_ALIASES"` occurs in the `fonts.alias` file, then a font can be referred to by its file name alone, without the path name or extensions. The X server will look in all the directories in your font path.

- A name you select.

You can specify what alias to use for referring to a font.

Any fonts not in the `fonts.alias` file must be referred to by the XLFD name.

When you edit a `fonts.alias` file, any X or font servers using that directory must be informed that they need to reread the file. To force an X server to read its `font.alias` files, run

```
xset fp rehash
```

To force a font server to read its `fonts.alias` files, run

```
kill -USR1 pid
```

where *pid* is the process ID of the font server.

Using Alias Names

For example, with this `fonts.alias` file and the `fonts/hp_roman8/75dpi` subdirectory of the system directory in the font search path:

```
"FILE_NAMES_ALIASES"
ellen *-adobe-courier-bold-r-normal-*-8-80-75-75-m-50-hp-roman8
```

then you can use any of the following commands to start a digital clock using this font:

- The "FILE_NAMES_ALIASES" entry lets you use just the file name, without the path or extension.

  ```
  xclock -digital -fn CourB08
  ```

- The alias name you specified.

  ```
  xclock -digital -fn ellen
  ```

- You can always specify the XLFD name, whether or not you have a `fonts.alias` file.

  ```
  xclock -digital -fn *-adobe-courier-bold-r-normal-*-8-80-75-75-m-50-hp-roman8
  ```

- You can specify enough of the XLFD fields to identify the font characteristics you want, and represent the rest with wildcards, with 14 dashes separating the fields. The X server selects the first font in its search path that matches the specification.

  ```
  xclock -digital -fn *-*-courier-bold-r-normal-*-8-*-*-*-*-*-hp-roman8
  ```

This is useful for vendor independence—you can have the same programs and default files on different vendors' computers, and customize by making the appropriate entry in the `fonts.alias` file.

Errors

If you get a default font or an error message (such as "can't make font ... ") when you request a font:

- Check the XLFD name for spelling.
- Check the XLFD name for inconsistencies. For instance, you should not specify both the PixelSize and PointSize for scalable typefaces. If you think there might be a conflict, set one of the parameters to an asterisk (*) and try again.
- Run `xlsfonts` to see if the font you requested is available to you.
- Run `xset q` to see if the directory containing the font you requested is in your font search path.
- Run `xset fp rehash` to be sure the X server is using the latest aliases and font paths.

Bitmapped Font Administration

Bitmapped fonts are included with the X Window System. They are located in the `fonts` subdirectories of `/usr/lib/X11`. You may use them as described in the following chapters without special installation or licensing steps.

Adding and Deleting Bitmapped Fonts

To add a bitmapped font:

1. If the font is not already in `.pcf` format, put it into the `.pcf` format using `bdftopcf`.
2. Compress the `.pcf` file using `compress`.
3. Copy the file into the desired directory.
4. Run `mkfontdir` to update the `fonts.dir` file for that directory.
5. If the directory is providing fonts to the X server, run `xset -fp rehash` to notify the X server of the changes. If the directory is providing fonts to a font server, run `kill -USR1` *pid*, to notify the font server of the changes.

To delete a bitmapped font:

1. Delete the font file.
2. Run `mkfontdir` to update the `fonts.dir` file for that directory.
3. If the directory is providing fonts to the X server, run `xset -fp rehash` to notify the X server of the changes. If the directory is providing fonts to a font server, run `kill -USR1` *pid*, to notify the font server of the changes.

Creating a fonts.dir file with 'mkfontdir'

The `mkfontdir` utility creates the `fonts.dir` file within a font directory.

The syntax for `mkfontdir` is:

`mkfontdir` *directory,directory,...*

where:

directory is a font directory. If no directory is given, the current directory is assumed.

Compiling BDF Fonts to PCF Fonts with 'bdftopcf'

X bitmapped fonts can be represented in several formats. A font's format is signified by the extension that appears after the font's file name:

.pcf	Portable binary font description file.
.pcf.Z	Compressed .pcf file.
.bdf	Plain text font description file.
.bdf.Z	Compressed .bdf file.
.bcf	Compressed .bdf file.
.snf	(Prior to X11R5) Non-portable binary font description file.
.snf.Z	(Prior to X11R5) Compressed .snf file.
.scf	(Prior to X11R5) Compressed .snf file.

Although all of these formats can be read by the X and font servers, the preferred representation for font storage are the .pcf and .pcf.Z formats. All bitmapped fonts shipped with HP-UX are shipped in the .pcf or .pcf.Z format.

The font compiler `bdftopcf` converts a font in bitmap distribution format (.bdf) into the .pcf format.

The syntax for `bdftosnf` is:

`bdftopcf [`*options*`]` *filename*

where *options* are:

`-p`*number*	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.
`-u`*number*	Force the scanline unit padding to 1, 2, 4, or 8.
`-l`	Specifies the output of `bdftopcf` to be least significant byte first.
`-L`	Specifies the output of `bdftopcf` to be least significant bit first.
`-m`	Specifies the output of `bdftopcf` to be most significant byte first.
`-M`	Specifies the output of `bdftopcf` to be most significant bit first.

`-t`	Expand glyphs in "terminal emulator" fonts to fill the bounding box.
`-i`	Don't compute ink metrics for "terminal emulator" fonts.
`-o` *pcf_file*	Specifies the name of the output .pcf file. If not specified, `bdftopcf` sends its output to stdout.
filename	Specifies the name of the .bdf font to be converted into portable compiled format.

The following example takes a bitmap distribution bitmap font file named `tmrm12b.bdf` and converts it to a compressed portable compiled format file:

```
bdftopcf tmrm12b.bdf | compress >timrm12b.pcf.Z
```

Scalable Typeface Administration

A font administrator is anyone who has purchased a font and wants to use it on a system. Scalable typefaces, unlike the X bitmap fonts, are licensed. Read your license carefully. The font administrator is responsible for ensuring that the font is used in a legal manner. The permissions for files and directories that relate to scalable typefaces have been carefully chosen to allow you to fulfill your responsibilities.

The font administrator has three main tasks:

- Install and delete scalable typefaces.
- License and unlicense devices to use typefaces.
- Add and delete character sets.

Each system is shipped with a core set of scalable typefaces installed in two different directories. Intellifont scalable typefaces from Agfa are installed in the directory `/usr/lib/X11/fonts/ifo.st`, and Type 1 scalable typefaces from Adobe are installed in the directory `/usr/lib/X11/fonts/type1.st`. You can list them by typing ...

```
xlsfonts -fn "*-0-0-0-0-*"
```

... as described earlier in this chapter.

Overview

There are four steps a font administrator must perform to make a scalable typeface ready for use. These steps are covered in more detail in the following sections.

1. Load the typeface into a directory on the target system ("Installing and Licensing Scalable Typefaces").
2. Load the character set if it is not already on the system ("Adding and Removing Character Sets"). Character sets are used by several different typefaces.

3. Run `stmkdirs` for that directory to notify the X server of the addition. ("Creating *.dir Files with 'stmkdirs'")
4. Add the license to that typeface for the system ("Adding and Removing Licenses with 'stlicense'").

An example of installation and removal of a typeface and its license is presented later in this chapter.

When you install a new scalable typeface, any running font servers that are using that directory need to be told to read the directory for the new font or fonts. You can do this by entering `kill -USR1` *pid*, where *pid* is the process ID of the font server. See the section on using the font server for more detailed information on font server configuration.

Installing and Licensing Scalable Typefaces

To install a scalable typeface onto a system:

1. Decide what directory will contain the new typefaces.
   - If you use the `fonts/ifo.st/` or `fonts/type1.st` subdirectories of `/usr/lib/X11`, you will have your fonts in centralized locations, but you need superuser capability to write in either of these directories.
   - If you create your own directory, you do not need superuser capability. If you create a new directory, be sure to:
     - give it the extension `.st`.
     - make it readable for the group `bin`.
     - configure the font server to include it in its font path.
2. Create a "typefaces" subdirectory to the `.st` directory, if one does not already exist. Install new fonts into this directory.

   For Intellifont format fonts:

   a. Copy the files containing the typeface into an empty temporary directory on the target file system.

   b. If your typefaces are contained on several flexible discs, load the entire contents of each disc into its own temporary directory *or* do these steps for each individual disc. Copy the entire contents of the disc, even if you want only one typeface from it.

Caution

If you copy all the discs into one directory, some files will be overwritten.

   - For HP-UX media, copy the files directly to the temporary directory.
   - For MS-DOS media, use the `doscp` utility to copy the files from a flexible disc drive to the temporary directory.

- If a PC is networked into your system, refer to the network documentation about how to copy files from the PC to the HP-UX system.

c. Run the `stload` utility on each temporary directory to convert the files into the proper format and place the typeface in the permanent directory you established in step 1.

d. If you loaded more typefaces than you wanted, remove the file(s) and run `stmkdirs` for that directory.

e. Delete the temporary directories used in step 2b.

For Type-1 format fonts:

Note

The font server can handle IBM-format Type 1 fonts, but it cannot read the Macintosh format.

a. Copy the desired scalable font files (extensions `.pfa` and `.pfb`) from the distribution media into the `typefaces` subdirectory. The suffixes *must* be lower case; the system does not recognize `.PFA` or `.PFB`.

b. Run `stmkdirs` for the `typefaces` subdirectory to add the new fonts to the database.

Once the file is loaded, typefaces can be made available to users through licensing. Refer to "Adding and Removing Licenses with 'stlicense'".

To delete a typeface from a system:

1. Remove all licenses for the product, using `stlicense`. For example

   ```
   stlicense -pr foo -fp /home/ellen/ifo.st "*"
   ```

   removes all licenses to product "foo" in the specified directory.

2. Remove the typeface files that are no longer being used (extensions `.ifo`, `.pfa`, and `.pfb`) from the `typefaces` subdirectory.

3. Run `stmkdirs` in the typefaces subdirectory to update the `fonts.dir` file.

4. Remove the product file from the `products` subdirectory.

Loading Scalable Typefaces with 'stload'

Use the `stload` utility to load into the system Agfa fonts that have been distributed in Agfa's FAIS distribution format.

The syntax for the `stload` utility is:

`stload` [*options*] [*directory\filespec*]

where:

directory\filespec	Required parameter specifying the name of the directory or filespec of the data to be loaded.
`-o` *path*	The name of the directory to which the the typeface outlines should be written. If this is omitted, the default is to the `/usr/lib/X11/fonts/ifo.st/typefaces` file.
`-fp` *path*	The name of the base directory under which `typefaces`, `metrics`, and `products` directories should be used.
`-p` *product-number*	Associates a product number with the newly-loaded typeface. Although this could be anything, it should reflect the product number on the package and media. The `stlicense` utility requires this product number.
`-list`	Prints a list of the data located in *directory*.
`-link`	Make links to the original *directory*, rather than copies.
`-sym`	Make symbolic links to the original *directory*.
-id[`,`*-id ...*]	Identifies one or more specific typefaces to be loaded.
`-tfm`	Updates `.tfm` files in the output directory.
`-dos`	Specifies that the typeface file is in DOS format. Normally, `stload` generates a typeface file in a format installed for use on HP-UX, and not available on DOS.
`-d` *mapdir*	Specifies the directory containing the symbol list map required by the `-to` option.
`-to` *format*	Specifies the symbol list that should be used for assigning character ID codes when loading FAIS data.
`-f` *libname*	Specifies the name of the library into which FAIS data should be loaded.
`-u`	Specifies that the `.dir` files *not* be updated.
`-v`	Specifies verbose mode.
`-h`	Requests help.

For example:

```
stload -fp new.st -p C2054#ABA -tfm -dos -v tempdir
```

Creating *.dir Files with 'stmkdirs'

Use the `stmkdirs` utility to create and maintain various configuration files that support the scalable typeface technology, including `fonts.dir` files, directories of character sets, and directories of metrics files.

This section describes how to use `stmkdirs` to maintain font directories. `stmkdirs` works much like `mkfontdir`, in that it creates the `fonts.dir` file containing a list of fonts in the current directory. But, `stmkdirs` recognizes Intellifont and Type 1 scalable fonts, while `mkfontdir` recognizes only bitmapped fonts. Therefore, you should use `stmkdirs` to build the `fonts.dir` file in any font directory that contains scalable fonts.

`stmkdirs` creates the `fonts.dir` file from the directory of font files.

`stmkdirs` [*options*] *directory*[, *directory,...*]

where the options are:

`-tfm`	For any Intellifont files in the target directories, build a TFM (Tagged Font Metrics) file in the specified destination directory.
±`m`	Requests that `fonts.dir` be generated including (`+`) or excluding (`-`) bitmap fonts.
±`o`	`fonts.dir` is generated including (`+`) or excluding (`-`) scalable fonts.
±`f`	`fonts.dir` is generated (`+`) including excluding (`-`) both scalable fonts and bitmap libraries, or not generated (`-`).
±`c`	Requests that `charsets.dir` be generated (`+`) or not generated (`-`).
`-b`	Suppresses creation of backup files.
`-h`	Prints help information on stout.
directory	is one or more directory names containing fonts.

For each directory listed, `stmkdirs` reads all the font files in that directory, putting file names and XLFD name into the `fonts.dir` file. Without a `fonts.dir` file, the the X and font servers cannot access font files in the directory.

Run `stmkdirs` after any fonts or charsets are added or deleted.

Adding and Removing Licenses with 'stlicense'

When you purchase a scalable typeface product, you receive a license agreement, outlining by who and how the typefaces in the product may be used. For instance, the terms may be that only one printer and one display may use the typefaces.

The `stlicense` utility helps administer the licenses. Fonts are available only to licensed devices.

The syntax for the `stlicense` utility is:

$$\texttt{stlicense}\ [\texttt{-fp}\ \textit{directory}]\left\{\begin{matrix}\texttt{-fn}\ \textit{typeface}\\ \texttt{-pr}\ \textit{product}\end{matrix}\right\}[\pm\textit{device}\ \ldots\ ]$$

where

`-fp` — The path of directories to search for the specified product or typeface. The default is `/fonts/ifo.st/` in `/usr/lib/X11`.

`-fn` — The *typeface* being licensed. The typeface is specified as an XLFD name. You need not use the whole XLFD name, just enough to uniquely identify the typeface. A product is identified by its name or product number.

`-pr` — The *product* being licensed.

±*device* — The *device* is specified in the form: `host:device`. The given typeface is added to or removed from the list of typefaces licensed for this device.

The host name `STSYSTEM` refers to all hosts served by this typeface directory.

The device name `DISPLAYS` refers to all displays running on the host.

The device name `PRINTERS` refers to all printers connected to the host.

If the machine is not specified, the default is the machine on which `stlicense` is running, and the device defaults to `DISPLAYS`.

nothing — If no devices are given, a list of devices that have licenses for the typeface is printed on the standard output. The list is grouped by system and individual device licenses.

The built-in typefaces are licensed at installation time to all displays and printers attached to the system (`STSYSTEM:DISPLAYS` and `STSYSTEM:PRINTERS`). For example,

```
stlicense -pr C2054#ABA +lj3
```

licenses the printer named lj3 to use the typeface product C2054#ABA. Since the machine is not specified, `stlicense` assumes the machine to be the one on which it is running.

```
stlicense C2054#ABA -pr -laserjp +laserkb
```

Adding and Removing Character Sets

Many Intellifont and Type 1 scalable fonts contain many characters, and can be used to create more than one character set. For example, both Intellifont and Type 1 scalable fonts can be used to build fonts using either ISO8859 encoding or HP roman-8 encoding. This section describes the management of character sets for scalable fonts.

Administering Character Sets for Intellifont Fonts. Character set definitions are stored in the `fonts/stadmin/ifo/charsets` subdirectory of `/usr/lib/X11`. as ASCII files with the extension `.sym`. The charsets directory is shipped from the factory with two popular character sets definitions:

- HP Roman 8
- ISO 8859-1 (also known as ECMA Latin 1)

These character sets are the only ones many applications need.

The `archive` subdirectory contains definitions for a number of additional character sets. These include character sets popular for PCs.

To enable one of the character sets in "archive":

1. Copy the desired character set (`.sym`) file from the `archive` subdirectory into the `charsets` directory. For example,

   ```
   cp /usr/lib/X11/fonts/stadmin/charsets/archive/pc8.sym ..
   ```

2. Run `stmkdirs` in the `charsets` directory to update the `charsets.dir` file. For example,

   ```
   stmkdirs /usr/lib/X11/fonts/stadmin/charsets
   ```

3. Run the following to notify the font server of the changes:

   `kill -USR1` *pid*

   where *pid* is the process ID of the font server.

To install a character set from the Type Director/DOS product, first run `stconv` on the `.sym` file to put it into a format that can be used on your workstation.

To delete a character set:

1. Remove the character set (`.sym`) file from the `charsets` directory. (It is still in the `archive` subdirectory if you need it later.)
2. Run `stmkdirs` with the `+c` option in the `charsets` directory to update the `charsets.dir` file.
3. Run the following to notify the font server of the changes:

   `kill -USR1` *pid*

   where *pid* is the process ID of the font server.

Administering Character Sets for Type 1 Fonts. Character set definitions for Type 1 fonts are stored in `/usr/lib/X11/fonts/stadmin/type1/charsets`. Files in this directory named `cp.`*`character_set`* define the character mapping for the desired character set. The two files shipped in this directory, `cp.iso8859` and `cp.hp-roman8`, define the character set mappings for ISO8859.1 and Roman-8 encoding.

To add or delete character set mappings for Type 1 fonts, you need to add or delete mapping files to this directory. The file names must be of the form `cp.`*`character_set`*, where *character_set* is the charset definition, containing one hyphen, to be used at the end of the font's XLFD name.

When a font server starts up or rereads its font directories in response to a signal, it uses the character sets defined in this directory to build its list of available font names.

Example: Installing and Licensing

This example shows installing and licensing an Intellifont typeface product called "COOOO#AAA". Path names are shown in full for clarity, you may not need to specify them in that detail. Assume that you have named a flexible disk drive device location `/dev/rdsk/2s1`.

"COOOO#AAA" is the product number on the box of the product. It comes on two flexible discs.

A new scalable typeface directory is to be created. It is owned by the font administrator, `/home/ellen`. A flexible disc drive is attached to the system at device location `/dev/rdsk/2s1`.

1. Copy each of the two discs into its own temporary directory.

```
mkdir /tmp/disc1
insert flexible disc 1 into the drive.
doscp /dev/rdsk/2s1/* /tmp/disc1
mkdir /tmp/disc2
insert flexible disc 2 into the drive.
doscp /dev/rdsk/2s1/* /tmp/disc2
```

2. Create a new directory for the scalable typeface and make it readable by the `bin` group. All other groups should have no access to the `.ifo` files.

```
mkdir /home/ellen/new.st
chacl "%.bin+r" /home/ellen/new.st
mkdir /home/ellen/new.st/typefaces
chacl "%.bin+r" /home/ellen/new.st/typefaces
```

3. Load the typefaces into the new directory. Note that this example includes the creation of `.tfm` files. If you have applications that utilize AutoFont Support, you will need them. Otherwise, save installation time and disc space by not requesting them.

```
stload -p COOOO#AAA -dos -v -fp /home/ellen/new.st -tfm /tmp/disc1
```

```
stload -p C0000#AAA -dos -v -fp /home/ellen/new.st -tfm /tmp/disc2
```

4. Make the new files readable by the `bin` group.

   ```
   chacl "%.bin+r" /home/ellen/new.st/typefaces/*"
   ```

5. Clean up the temporary directories.

   ```
   rmdir /tmp/disc1
   rmdir /tmp/disc2
   ```

6. To make the fonts available to a font server, edit the font server's `config` file to add the new font path. For example, you can append:

   ```
   ,/home/ellen/new.st
   ```

   to the "catalogue =" entry in the `config` file. Then force the font server to reread its `config` file by typing:

   ```
   kill -USR1 pid
   ```

   where *pid* is the font server's process id.

7. Before this product can be used, it must be licensed. For this example, the license in the product stipulates that the typefaces can be used for up to three printers and any number of displays connected to the system.

   ```
   stlicense -fp /home/ellen/new.st -pr C0000#AAA +STSYSTEM:DISPLAYS \
      +mysystem:laser1 +mysystem:laser2 +mysystem:laser3
   ```

   Notice that although the printers are listed individually, the displays are grouped by the shortcut `STSYSTEM:DISPLAYS`. `mysystem` is one of the hosts covered by `STSYSTEM`.

   If you now wanted `mysystem:laser4` to be licensed, you have to remove the license for one of the other printers, since you are only allowed up to three printers.

   ```
   stlicense -fp /home/ellen/new.st -pr C0000#AAA -mysystem:laser3 \
                                                  +mysystem:laser4
   ```

When the product is no longer needed, remove it from the system.

1. Remove all licenses to the product.

   ```
   stlicense -fp /home/ellen/new.st -pr C0000#AAA "-*"
   ```

2. Remove the typeface files (`.ifo`). The list of files to be removed is in `/home/ellen/new.st/products/C0000#AAA`.

   ```
   rm /home/ellen/new.st/typefaces/12345678.ifo
   rm /home/ellen/new.st/typefaces/22345678.ifo
   ⋮
   ```

3. Update the `fonts.dir` in the `typefaces` subdirectory.

   ```
   stmkdirs /home/ellen/new.st/typefaces
   ```

Scalable Typefaces File Structure

This section describes the default scalable font directories (font catalogs). There can be other font catalogs, but each must have the `.st` extension and structure described here. In addition, each must be on the font path.

The directories described here are subdirectories of `/usr/lib/X11`.

Scalable Font Directories

The `fonts/ifo.st` and `fonts/type1.st` are the default font catalogs. They contain typeface files, licensing, and metrics information.

Licenses Subdirectory. The `licenses` subdirectory contains files with licensing information for each host, display, and system.

It contains a `hosts.dir` file, which is a cross-reference between the actual host name and the directory containing license information about that host. One host subdirectory is `STSYSTEM`, which is for system-wide licenses. There are separate subdirectories for each host on the system.

Within each host subdirectory, there are subdirectories for each device (`DISPLAYS` is always one). Within these directories there are `fonts.dir` and `fonts.alias` files as described elsewhere in this manual.

Metrics Subdirectory. The `fonts/ifo.st/metrics` directory contains metrics for the fonts and scalable typefaces that are not loaded on the system. This is the recommended location for the `.tfm` files for Intellifont fonts, and for `.afm` files for Type 1 fonts.

Products Subdirectory. Each product that has been installed has a file cross-referencing the font file name and the XLFD name used to refer to it. The core fonts are in the `builtin` file.

Typefaces Subdirectory. The `typefaces` subdirectory contains the typeface files. The Intellifont files have a `.ifo` extension. Type 1 typeface files have a `.pfa` or `.pfb` extension. In addition, there is a `fonts.dir` file for each typeface directory.

Administrative Directories

The `fonts/stadmin/ifo` directory contains `typefaces.dir`, which provides a cross-reference between the typeface ID and the XLFD name for Intellifont fonts.

The `fonts/stadmin/ifo/charsets` subdirectory contains valid character sets for Intellifont fonts. These files have a `.sym` extension and are in the same format at those for TypeDirector/DOS 3.0. A `charsets.dir` file provides a cross-reference between the file name and the character set name. Non-active character sets are contained in the subdirectory `archive`, with its own `charsets.dir` file. To make an inactive character set active, copy it from the `archive`

subdirectory, and update the `charsets.dir` file by running `stmkdirs` on that directory.

The `fonts/stadmin/type1/charsets` subdirectory contains valid character sets for Type 1 fonts. These files are all named `cp.`*character_set*, and are used to provide a mapping between internal Type 1 character names and standard encodings such as ISO8859.1 and Roman-8.

Using 'stmkfont' and 'stconv'

Two additional support utilities are provided for use with Intellifont outline fonts. The next two sections describe `stmkfont`, a utility for generating a variety of bitmap formats from Intellifont outlines, and `stconv`, a utility for manipulating Intellifont symbol set files.

Making Bitmapped Fonts from Scalable Typefaces with 'stmkfont'

The `stmkfont` utility produces bitmapped fonts in a variety of formats from an outline specified by an XLFD name. `stmkfont` can create bitmap fonts in the following formats:

bdf	Bitmap Distribution Format
PCL	Printer Command Language (for HP LaserJet printers)
PCLEO	Printer Command Language Encapsulated Outlines (for HP LaserJet III printers).
IFO	Intellifont outline.
TFM	HP Tagged Font Metric for metrics pertaining to HP LaserJet printer scalable typefaces.

The syntax for `stmkfont` is:

`stmkfont` [*options*] *xlfdname*

where the options are:

`-d1`*path*	Specifies the primary database tree path (default is `fonts/ifo.st`).
`-d2` *path*	Specifies the secondary database tree path (default is `fonts/stadmin`).
`-dv` *device*	Specifies the device for which the font is to be made.
`-cp` *path*	Specifies the charset path (default is `charsets`).
`-cf` *file*	Specifies the charset file (default is to derive it from the XLFD name.
`-nf` *file*	Specify a new name for `fonts.dir`.
`-ns` *file*	Specify a new name for `charsets.dir`.
`-nt` *file*	Specify a new name for `typefaces.dir`.
`-nv` *name*	Specify an environment variable to use instead of STPATH.
`-o` *outfile*	Specifies output file (default is stdout)

`-f` *format*	Specifies the output format (BDF (default), PCL, or PCLEO).
`-I`	Send completion status information to stderr.
`-P`	Send 1% progress dots to stderr.
`-C`	Send catalog of XLFD/symbol set combinations to stderr.
`-T`	Bypass intermediate tempfile, write to output directly.
`-V`	Send fully qualified XLFD name to stderr, then quit.
`-v`	Send fully qualified XLFD name to stderr, then continue.
`-w`	Suppress bitmaps, restrict output to header and trailer only.
`-q`	Suppress error messages (quiet mode).
xlfdname	This parameter is required. Since both the XLFD name and parameters start with a dash (`-`), then `stmkfont` assumes the last arguments is the XLFD name.

The XLFD name must not contain any blanks. If it does, enclose the entire string in quotes (`"`). Empty fields and wildcards are permitted.

For example,

```
stmkfont -o myfont "-agfa-cg century schoolbook-normal-r-normal-*--240---p-150-*-roman8"
```

Converting Map Formats with 'stconv'

The `stconv` utility converts symbol set maps (`.sym` files) from one symbol set to another. Output is always to `stdout`.

The syntax for `stconv` is:

`stconv` *infile* [`-hmq`][`-d` *mapdir*][`-to` *format*]

where:

infile	Name of the `.sym` file to be converted.
`-d` *mapdir*	Specifies the name of the directory containing the symbol conversion list. This directory should contain the file `acg.hpmsl` and any optional additional symbol set maps. The default is `fonts/stadmin/ifo/charsets` in `/usr/lib/X11`.
`-to` *format*	Specifies the new symbol list format. The default is `hpmsl`. To generate a symbol set for Agfa's character codes, specify `-to ACG`.
`-m`	Lists the conversion map.
`-q`	Run quietly.

`-h` Requests help.

For example,

```
stconv -to ACG roman8.sym
```

reads the HPMSL symbol map `roman8.sym`, and writes the ACG version to stdout.

7

The Window Manager

The OSF/Motif Window Manager (`mwm`) is an X11 client that manages the appearance and behavior of objects on the root window. You control `mwm` and its management functions using a mouse, keyboard, and a functional window frame. Additionally, `mwm` has a root menu to assist you in the control of the root window.

Chapter 4 explains how to *use* windows. This chapter explains how to customize them.

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

- Starting and stopping `mwm`.
- Setting `mwm` resources using `.mwmrc`
- 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 focus policies.

Starting and Stopping the Window Manager

The OSF/Motif Window Manager (`mwm`) is an X11 client that manages the appearance and behavior of objects on the root window. You control `mwm` and its management operations using a mouse, a keyboard, and a functional window frame. Additionally, `mwm` has a root menu to assist you in the general control of the root window.

The OSF/Motif 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, `mwm` is started from `sys.x11start` in `/usr/lib/X11`.

The syntax for `mwm` is as follows:

```
mwm [options]
```

where *options* are:

`-display` *host:display.screen* Specifies the screen to use.

`-xrm` *resourcestring*	Specifies using the named resource on starting.
`-multiscreen`	Causes `mwm` to manage all screens on a display. The default is to manage only a single screen.
`-name` *name*	Uses *name* to retrieve resources.
`-screens` *name [name ...]*	Gives the resource names for the screens managed by `mwm`. The names are separated by spaces.

The following line in `.x11start` in your home directory starts `mwm`.

```
mwm $@ &
```

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

Declaring Resources

The `mwm` client receives configuration information from three resource files:

- `sys.Xdefaults` in `/usr/lib/X11` or `.Xdefaults` in your home directory.

  Contains X resources.

- `system.mwmrc` in `/usr/lib/X11` or `.mwmrc` in your home directory.

  Menus, key bindings, and button bindings.

- `app-defaults/Mwm` in `/usr/lib/X11`.

  X resources for `mwm` only.

  This file cannot be changed. However, you can copy information from that file, modify it, and then add it to your personal resource.

If you modify these files, you can use either method of specifying personal resources: changing the RESOURCE_MANAGER property or modifying the `.Xdefaults` file. Both methods are covered in chapter 5.

The syntax you use differs depending on whether you want the resource to control an element or that element *for a particular object*.

The syntax for `mwm` resources is:

$$\texttt{Mwm*}\left[\left\{\begin{array}{l}\textit{clientname clientclass}\\ \texttt{defaults}\end{array}\right\}\right]\texttt{*}\textit{resource}\texttt{:}\ \ \textit{value}$$

Use nothing between "Mwm" and the resource name if you want the resource applied to all clients for which you don't otherwise specify a value. Some resources make sense only at this level, such as the focus policy ones. Use *clientclass* to apply the resource to a specific class of clients. Use *clientname* to apply the resource only to a specific

instance of a client named using the client's `name` resource. Use `defaults` when you want the default value used.

For example, if you want the general appearance of the clients in your environment to be SteelBlue and VioletRed, but want your menus to be different, you could use the following lines in your personal resources.

```
Mwm*background:         SteelBlue
Mwm*foreground:         VioletRed
Mwm*activeBackground:   VioletRed
Mwm*activeForeground:   SteelBlue

Mwm*menu*background:    SkyBlue
Mwm*menu*foreground:    White
```

Or, 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 personal resources:

```
Mwm*HPterm*iconImage: /home/yourusername/Bitmaps/face.bits
Mwm*iconDecoration: label activelabel
```

Frames

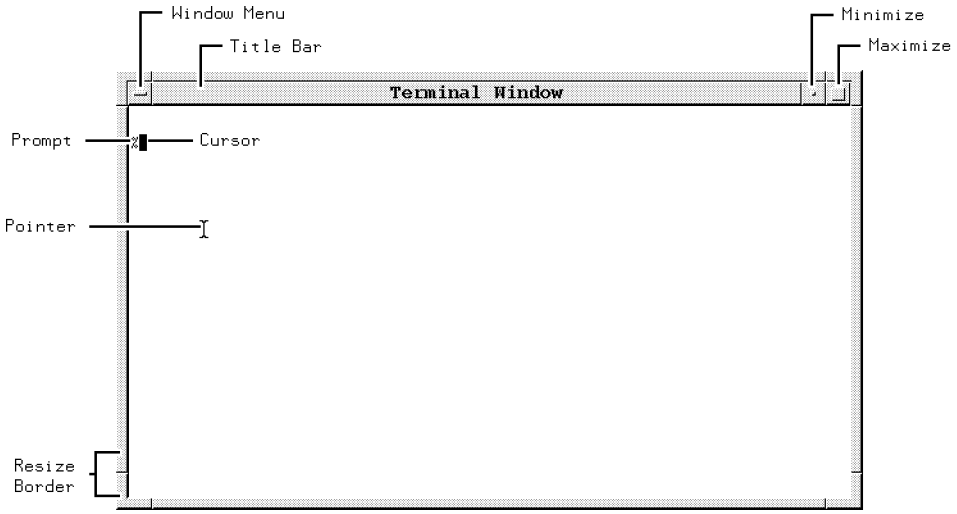
You can control the general appearance of the window frames in your environment with your personal resources specifications.

Parts of a Window Frame

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.

Additionally, you can control what parts of the frame are displayed.



Frame Elements

Customizing the Window Frames

You can specify what frame components you want to appear on windows:

- 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 access the functionality of any decoration you remove by binding its functions to mouse buttons or to key presses, as explained in "Mouse Button Bindings" later in this chapter.

Valid Window Frame Elements

Frame Element	Description
all	Include all frame elements (default value).
none	Include no window frame elements.
±border	Window border.
±maximize	Maximize button (includes title bar).
±minimize	Minimize button (includes title bar).
±none	Include no window frame elements.
±resizeh	Resize border handles (includes border).
±menu	Window menu button (includes title bar).
±title	Title bar.

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 or has no sign preceding it, 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.

For example, you may want a border with only a title bar and window menu button around a particular `hpterm` window started as `hpterm -name hp850`.

```
Mwm*hp850*clientDecoration: +menu
```

Or you could remove the title bar from all transient windows by adding the following line in your personal resources specification:

```
Mwm*transientDecoration: –title
```

Coloring Window Frame Elements

You can use any of the standard X11 colors listed in the `rgb.txt` file in `/usr/lib/X11`. to color frame elements. In addition, you can create your own colors using hexadecimal values (see "Color Resources" in chapter 5).

The following table lists the individual elements of inactive and active window frames, and the resources that control their color, for `mwm`.

The default settings provide a 3-D visual effect without you having to specify the exact colors for every frame element.

Window Frames Resources for a Color Display

To color this ...	Use this resource ...	The default value is ...
Background of inactive frames.	`background`	LightGrey
Left and upper bevel of inactive frames.	`topShadowColor`	Lightened `background` color
Right and lower bevel of inactive frames.	`bottomShadowColor`	Darkened `background` color
Foreground (title bar text) of inactive frames.	`foreground`	Darkened `bottomShadowColor`
Background of the active frame.	`activeBackground`	CadetBlue
Left and upper bevel of the active frame.	`activeTopShadowColor`	Lightened `activeBackground` color
Right and lower bevel of the active frame.	`activeBottomShadowColor`	Darkened `activeBackground` color
Foreground (title bar text) of the active frame.	`activeForeground`	Darkened `activeBottomShadowColor`

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 background color is used to generate colors for the top and bottom shadow elements so that a 3-D effect is achieved.

```
Mwm*foreground: Maroon
Mwm*background: Gray
```

Tiling Window Frames With Pixmaps

A **pixmap** can be used to create shades of colors. Each pixmap is composed of tiles. A **tile** is a rectangle that provides a surface pattern or a visual texture by "mixing" the foreground and background colors into a color pattern.

Tiling Window Frames with Window Manager Resources

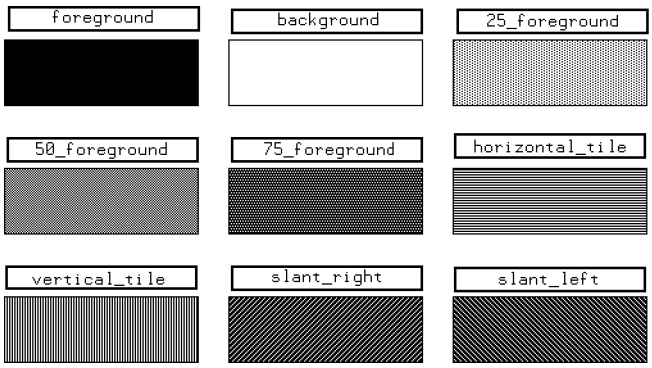
To tile this ...	Use this resource ...	The default for color displays is ...
Background of inactive frames.	`backgroundPixmap`	NULL
Right and lower bevels of inactive frames.	`bottomShadowPixmap`	NULL
Left and upper bevels of inactive frames.	`topShadowPixmap`	NULL
Background of the active frame.	`activeBackgroundPixmap`	NULL
Right and lower bevels of the active frame.	`activeBottomShadowPixmap`	NULL
Left and upper bevels of the active frame.	`activeTopShadowPixmap`	NULL

The following table lists the acceptable values for pixmap resources:

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:



Valid Tile Values

Matting Clients

A **matte** is a 3-D border just inside the window between client area and window frame.

The following table lists matte elements and the resources that control their color.

Coloring Window Frames with Window Manager Resources

To color this ...	Use this resource ...	The default value is ...
Width of matte	`matteWidth`	0 (no matte)
Matte background.	`matteBackground`	`mwm` background
Left and upper bevel of matte.	`matteTopShadowColor`	Lightened `matteBackground` color.
Right and lower bevel of matte.	`matteBottomShadowColor`	Darkened `matteBackground` color.
Matte foreground.	`matteForeground`	Darkened `matteBottonShadowColor`.
Matte right and lower bevels.	`matteBottomShadowPixmap`	client bottom shadow color
Matte left and upper bevels.	`matteTopShadowPixmap`	client top shadow color

The values to use for tiling mattes are shown in "Tiling Window Frames With Pixmaps".

For example, you could place a different matte around all instances of `hpterm` and `xterm` windows by including the following lines in your personal resources specifications:

```
Mwm*HPterm*matteWidth:        10
Mwm*HPterm*matteBackground:   SkyBlue
Mwm*XTerm*matteWidth:         10
Mwm*XTerm*matteBackground:    Tan
```

Frame Resources For Monochrome Displays

If `mwm` determines that the monitor is monochrome, and no color resources are specified for frame elements, `mwm` uses defaults appropriate for monochrome displays. `Mwm*background` and `Mwm*activeBackground` are set to White. The following table lists the frame elements, resources, and defaults for monochrome monitors.

Window Frame Resource Values for Monochrome Monitors

The background is ...	For this resource ...	The default value is ...
White	`topShadowColor`	White
White	`bottomShadowColor`	Black
White	`foreground`	Black
White	`topShadowPixmap`	foreground
White	`activeBackgroundPixmap`	foreground
White	`activeTopShowdowPixmap`	50_foreground

The `sys.Xdefaults` file contains a set of entries that provides a more attractive window shading for monochrome displays. These entries start with `mwm_bw`, and require that you start `mwm` with the name `mwm_bw`. To do this, edit the following line in `.x11start`:

```
mwm & #Starts the mwm window manager
```

to read:

```
mwm -name mwm_bw & #Starts the mwm window manager
```

You must restart X11 in order for this change to take effect.

When you start the window manager with a new name, it will no longer see resources of the form `mwm*`*resource*. It will see the class resources `Mwm*`*resource*.

Controlling Window Size and Placement

The following table lists window manager resources enabling you to refine your control over the size and placement of windows.

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 windows beyond the size specified in `maximumClientSize`.	`limitResize`	False
The maximum size of a client window set by 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.

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.

```
Mwm*XLoad.maximumClientSize: 128×96
```

Controlling Focus Policies

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 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:

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

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.

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:

```
Mwm*keyboardFocusPolicy: pointer
```

Specifying a Different Font for the Window Manager

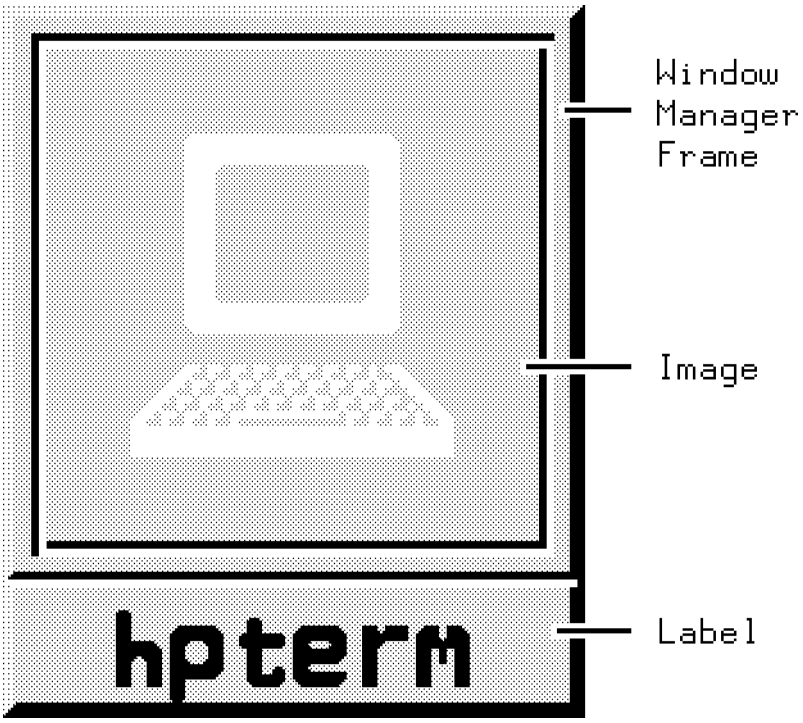
The default font for the text of the OSF/Motif Window Manager is the `fixed` font. However, you can use the `fontList` resource to specify a different font if you desire. The `fontList` resource can use any valid X11 font name as its value. For more information about fonts, see chapter 6.

Displaying Titles in Local Languages

If you want to display titles and icon names in languages other than English, you must set the `LANG` environment variable to the apporpriate language, and ensure that a font set appropriate to that language is set in the `mwm fontList` resource before the window manager is started.

Working with Icons

Icons provide a handy way to straighten up a cluttered workspace.



The Parts of an Icon

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, listed in order of precedence:

user	You can specify an icon image using the `iconImage` resource.
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.
default	The window manager will use its own built-in default icon image if an image is not specified elsewhere.

The window manager searches for an icon image in the order shown above. It stops searching when it finds the first image that meets the selection criteria.

The resource `useClientIcon` lets you interchange the precedence of user-supplied icon images and client-supplied icon images. The default value is "False." When the resource is set to "True," client-specified icon images have precedence over user-supplied icon images.

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:

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

The following table lists the icon placement schemes available to you:

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`

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 personal resource specifications:

```
Mwm*iconPlacement: top right     Specifies the placement scheme.
Mwm*iconAutoPlace: True          Specifies automatic placement.
```

Controlling Icon Appearance and Behavior

`mwm` offers you a number of resources to control the specific appearance and behavior of icons.

Selecting Icon Decoration

Using the `iconDecoration` resource, you can select exactly what parts of an icon you want to display:

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. `mwm` has both default sizes as well as maximum and minimum allowable sizes.

The Maximum and Minimum Sizes for Icon Images

	Maximum Size	Minimum Size
Default	50×50 pixels	32×32 pixels
Allowable	128×128 pixels	16×16 pixels

How the window manager treats an icon depends on the size of the image in relation to the maximum and minimum sizes.

Image 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:

Controlling Icon Image Size

To specify this ...	Use this resource ...
Maximum size of an icon image.	`iconImageMaximum`
Minimum size of an icon image.	`iconImageMinimum`

Bear in mind that the overall width of an icon is the image width *plus* border padding and the image height is the icon 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.

There are two resources that tell the window manager where custom icons are located:

- The `iconImage` resource specifies the bitmap for a particular icon image. Its value is the path to the file containing the bitmap. If this resource is specified, it overrides any client-specified images.
- The `bitmapDirectory` resource causes 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`.

Coloring and Tiling Icons

A number of resources enable you to specify the colors of icon elements.

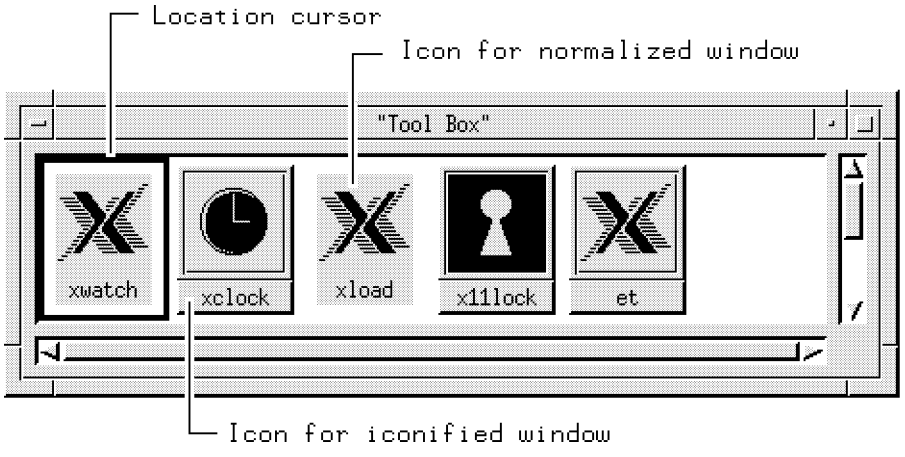
Coloring and Tiling Icon Resources

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`
Right and lower bevels of an icon image.	`iconImageBottomShadowPixmap`
Left and upper bevels of an icon image.	`iconImageTopShadowPixmap`

Default values for these resources are the icon's bottom and top shadow pixmaps specified using the `bottomShadowPixmap` and `topShandowPixmap` resources set by the entries `Mwm*icon*`*resource* or `Mwm*`*resource*.

Using the Icon Box to Hold Icons

The icon box allows you to use an icon box to contain icons, rather than having stand-alone icons on the workspace.



Icon Box

The icon box is a scrollable window that displays icons in a grid (rows and columns). Icons in the icon box do not overlap. If there are icons that cannot be displayed in the visible part of the icon box,

you can scroll to see the icons. The sliders within the scroll bars show the extent of the icon grid that is visible.

The icon box can be minimized (iconified) just like any other window. If the icon box is minimized, it is placed into the icon grid on the workspace.

Specifying the Icon Box

Several resources specify whether an icon box is used, define its geometry and location, and specify its name (for looking up resources) and title.

- The `useIconBox` resource specifies whether or not an icon box is used. A value of of "True" places icons in an icon box. The default value of "False" places icons on the root window.
- The `iconBoxGeometry` resource sets the initial size and placement of the icon box. If the `iconBoxGeometry` resource is used, the largest dimension of the size determines if the icons are placed in a row or a column. The default policy is to place icons in a row going from left to right, top to bottom.

The value of the `iconBoxGeometry` resource is a standard window geometry string with the following syntax:

$$\text{- } Width \times Height \left[\pm\ x \pm\ y \right]$$

If x and y are not provided, the icon box is placed at $+0-0$.

The actual size of the icon box window depends on the `iconImageMaximum` (size) and `iconDecoration` resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high.

- The `iconBoxName` resource specifies the name that is used to look up icon box resources. The default name is "iconbox."
- The `iconBoxTitle` resource specifies the name that is used in the title area of the icon box frame. The default name is "Icons."

For example, the following line specifies that icons will be placed in an icon box:

```
Mwm*useIconBox:  True
```

Controlling the Appearance of Icon Boxes

The icon box is displayed in a standard window management client frame. Client-specific resources for the icon box can be specified using "iconbox" as the client name.

```
Mwm*iconbox*resource: value
```

Resources that can be used with the icon box to change its appearance are:

- `iconDecoration`.
- The `mwm` resources dealing with mattes and icon appearance. (The icon appearance resources affect the icon displayed when the icon box is minimized.)

The Icon Box Window Menu

The window menu for the icon box differs from the standard window menu in that it does not contain the "Close" selection. In its place is the "PackIcons" selection, which shifts icons to fill empty spaces in the icon placement grid so that the icons appear in neat, complete rows.

Controlling Icons in the Icon Box

Every client window that can be iconified has an icon in the icon box, even when the window is in the normal state. The icon for a client is put into the icon box when the client becomes managed by the window manager, and is removed from the icon box when the client stops.

Icons for windows in the normal (open) state are visually distinct from icons for windows that are iconified. Icons for windows that are iconified look like stand-alone icons. Icons for windows that are in the normal state appear flat and are optionally grayed-out. The value of "True" for the `fadeNormalIcon` resource grays out icons for normalized windows. The default value is "False."

The text and image attributes of icons in icon boxes are determined in the same way as for stand-alone icons, using the `iconDecoration` resource.

A standard "control" location cursor is used to indicate the particular icon in the icon box to which keyboard actions apply. The location cursor is an unfilled rectangle that surrounds the icon.

Icons contained in the icon box can be manipulated with the mouse and from the keyboard. Mouse button actions apply whenever the pointer is on any part of the icon.

Controlling Icons in the Icon Box With a Mouse

If you want to ...	Do this ...
Select an icon.	Press button 1.
Normalize (open) an iconified window.	Double-click mouse button 1.
Raise a normalized window to the top of the stack.	Double-click mouse button 1.
Move an icon within the icon box.	Drag button 1.

To manipulate an icon from the keyboard, make the icon box the active window and use the arrow keys to traverse the icons in the icon box. Pressing Return does the default action for the selected icon: for an icon of a normalized window, the window is raised; for an icon of an iconified window, the window is normalized. The arrow keys move the focus around the icons that are visible. The Tab key

moves the keyboard input focus around the box in this order: icons, horizontal scroll bar, vertical scroll bar, icons. Shift Tab moves the focus in the opposite direction.

Managing Window Manager Menus

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

Default Menus

The OSF/Motif Window Manager comes with two default menus:

Default Window Menu

The default window menu is built into `mwm`. For reference, a copy of its contents are placed in `.mwmrc`.

```
Menu DefaultWindowMenu
{
    "Restore"   _R      Alt<Key>F5      f.normalize
    "Move"      _M      Alt<Key>F7      f.move
    "Size"      _S      Alt<Key>F8      f.resize
    "Minimize"  _n      Alt<Key>F9      f.minimize
    "Maximize"  _x      Alt<Key>F10     f.maximize
    "Lower"     _L      Alt<Key>F3      f.lower
    no-label                            f.separator
    "Close"     _C      Alt<Key>F4      f.kill
}
```

By default, the window menu displays when you do the following operations:

- Press button 1 on a window frame's window menu button.
- Press button 3 anywhere on a window frame.
- Press Shift Esc with the keyboard focus set to a window.

The `windowMenu` resource must be set in order to replace the `DefaultWindowMenu` with a different menu.

Default Root Menu

The default root menu is specified in the same files by the following lines:

```
Menu RootMenu
{
 "Root Menu"    f.title
 "New Window"   f.exec "hpterm &"
 "Start Clock"  f.exec "xclock -geometry 100x90-1+1 &"
 "Start Load"   f.exec "xload -geometry 150x90-130+1 &"
 "Shuffle Up"   f.circle_up
 "Shuffle Down" f.circle_down
 "Refresh"      f.refresh
  no-label      f.separator
 "Restart..."   f.restart
}
```

By default, the root menu displays when you press button 3 on the root window.

Modifying Menus

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 window menu the same.

All window manager menus, regardless of the mechanism that calls them to the screen, have the same syntax.

Menu Syntax

```
Menu MenuName
{
    selection1 [mnemonic]  [accelerator]  function [argument]
    selection2 [mnemonic]  [accelerator]  function [argument]
    selection3 [mnemonic]  [accelerator]  function [argument]
                                  ⋮
    selection* [mnemonic]  [accelerator]  function [argument]
}
```

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.

The `f.title` function creates a menu title, and automatically places a separator above and below the title.

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.

Mnemonics and Accelerators

You have the option of using a mnemonic and accelerator with a menu selection. A mnemonic is specified using the syntax:

`mnemonic = _`*character*

An accelerator is specified using keyboard binding syntax described later in this chapter (see "Keyboard Binding Syntax").

Functions

Each function operates in one or more of these contexts:

root	Operates the function when the root window is selected.
icon	Operates the function when an icon is selected.
window	Operates the function when a client window is selected.

Each function is triggered by one or more of these devices:

Button	Button binding (mouse).
Key	Key binding.
Menu	Window manager menu.

Most contexts and devices are valid for most functions. Occasionally, a context or device doesn't make sense for a particular function. Any selection that uses an invalid context, an invalid function, or a function that doesn't apply to the current context is grayed out.

Function Names, Contexts, and Devices

The following table lists the valid functions, contexts, and devices.

Valid Window Manager Functions

Functions		Contexts			Devices		
Name	Description	Root	Icon	Window	Button	Key	Menu
f.beep	Causes a beep to sound.	X	X	X	X	X	X
f.circle_down	Puts window on bottom of stack.	X	X	X	X	X	X
f.circle_up	Puts window on top of stack.	X	X	X	X	X	X
f.exec	Uses `/usr/bin/sh` to execute a command.	X	X	X	X	X	X
f.focus_color	Sets colormap focus when colormap focus policy is explicit.	X	X	X	X	X	X
f.focus_key	Sets keyboard input focus when keyboard focus policy is explicit.	X	X	X	X	X	X
f.kill	Terminates a client's connection to server.		X	X	X	X	X
f.lower	Lowers a window to bottom of stack.		X	X	X	X	X
f.maximize	Enlarges a window to its maximum size.		X	X	X	X	X
f.menu	Associates a menu with a selection or binding.	X	X	X	X	X	X
f.minimize	Changes a window into an icon.			X	X	X	X
f.move	Enables the interactive moving of a window.		X	X	X	X	X

Valid Window Manager Functions (continued)

Functions		Contexts			Devices		
Name	Description	Root	Icon	Window	Button	Key	Menu
f.next_cmap	Installs the next colormap in the window with the colormap focus.	X	X	X	X	X	X
f.next_key	Sets keyboard focus policy to the next window/icon in the stack.	X	X	X	X	X	X
f.nop	Does no function.	X	X	X	X	X	X
f.normalize	Displays a window in normal size.		X	X	X	X	X
f.pack_icons	Packs icons rows in the root window or icon box.	X	X	X	X	X	X
f.pass_keys	Toggles between enabling and disabling processing of key bindings.	X	X	X	X	X	X
f.post_wmenu	Posts the window menu	X	X	X	X	X	
f.prev_cmap	Installs the previous color map in the window with the colormap focus.	X	X	X	X	X	X
f.prev_key	Sets the keyboard input focus to the next window/icon in the stack.	X	X	X	X	X	X

Valid Window Manager Functions (continued)

Functions		Contexts			Devices		
Name	Description	Root	Icon	Window	Button	Key	Menu
f.quit_mwm	Terminates OSF/Motif Window Manager, but not X.	X			X	X	X
f.raise	Lifts a window to the top of the window stack.		X	X	X	X	X
f.raise_lower	Raises a partially concealed window; lowers an unconcealed window.		X	X	X	X	X
f.refresh	Redraws all windows.	X	X	X	X	X	X
f.refresh_win	Redraws a client window.			X	X	X	X
f.resize	Enables you to interactively resize a window.			X	X	X	X
f.restart	Restarts the OSF/Motif Window Manager.	X			X	X	X
f.send_msg	Sends a client message.		X	X	X	X	X
f.separator	Draws a line between menu selections.	X	X	X			X
f.set_behavior	Restarts `mwm` with CXI or custom behavior.	X	X	X	X	X	X
f.title	Inserts a title into a menu at the specified position.	X	X	X			X

Changing the Menu Associated with the Window Menu Button

The `windowMenu` resource lets you change the *menu* displayed when you press button 1 on the window menu button.

For example, you would place the following line in your personal resource specifications to associate a menu named `EditMenu` with an `hpterm` window started as `hpterm -name hp850`.

```
Mwm*hp850*windowMenu:    EditMenu
```

Mouse Button Bindings

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 `.mwmrc` file that associates a button operation with a window manager function.

Default Button Bindings

The OSF/Motif Window Manager comes with the following built-in button bindings.

Built-In Button Bindings

Location of Pointer	Behavior
Window menu button	Pressing button 1 displays the window menu. This behavior can be modified by the `wMenuButtonClick` resource.
Window menu button	Double-clicking button 1 closes the window. This behavior can be modified by the `wMenuButtonClick2` resource.
Minimize button	Clicking button 1 minimizes the window.
Maximize button	Clicking button 1 maximizes the window.
Title bar	Dragging button 1 moves the window.
Window or icon	Pressing button 1 gives it keyboard focus.
Resize border	Dragging button 1 resizes the window.
Icon	Clicking button 1 displays the icon window menu. This behavior can be modified by the `iconClick` resource.
Icon	Double-clicking button 1 normalizes the window.
Icon	Pressing button 1 moves the icon.

These bindings are fixed—they cannot be *replaced* by other bindings. However, you can *add* to some of them (see "Modifying Button Bindings and Their Functions.") For example, you can specify an additional function for double-clicking button 1 in an icon, but the double click will also normalize the window.

Mwm provides an additional default binding that can be deleted or replaced:

Additional Button Bindings

Locaton of Pointer	Behavior
Icon or Frame	Pressing button 1 raises the window or icon.

This binding is listed in the following section of the .mwmrc file.

```
Buttons DefaultButtonBindings
{
 <Btn1Down>  icon|frame  f.raise
}
```

The binding can be removed or altered by deleting or editing the line that begins with <Btn1Down>. (In order for the editing to have an effect, the buttonBindings resource must be set to DefaultButtonBindings, and you must restart the window manager.)

Modifying Button Bindings and Their Functions

You can modify the button bindings section of your .mwmrc 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]
}
```

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 or window frame.
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.

`app`	Operates the function when the button is activated in a client window (excludes window frames).

Modifying Button Bindings

Button bindings can be modified by:

- Editing the `DefaultButtonBindings` section of `.mwmrc`.
- Making a new button binding set.

To create a new button binding set:

1. Edit the `.mwmrc` file to include a new key binding set with a unique name.
2. Set the `buttonBindings` resource in your `.Xdefaults` file to the new name.

Modifying Button Click Timing

The OSF/Motif 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. The default is 500 (milliseconds).

Keyboard Bindings

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.
- Alt (Meta or Extend Char).
- Tab.
- Ctrl.
- Lock.

Default Key Bindings

The OSF/Motif Window Manager comes with the following default key bindings.

OSF/Motif Window Manager Default Keyboard Bindings

When the keyboard focus is:	Press:	What this does is:
Window or icon	Shift Escape	Displays window menu.
Window or icon	Alt space	Displays window menu.
Window, icon, or none	Alt Tab	Switches keyboard focus to the next window or icon.
Window, icon, or none	Alt Shift Tab	Switches keyboard focus to the previous window or icon.
Window, icon, or none	Alt Escape	Switches keyboard focus to the next window or icon.
Window, icon, or none	Alt Shift Escape	Switches keyboard focus to the previous window or icon.
Window	Alt F6	Switches keyboard focus to the next window or icon, including transient windows.
Window, icon, or none	Alt Ctrl Shift !	Restart `mwm` with default or custom behavior.

These keyboard bindings are listed in the following lines in `system.mwmrc` and `.mwmrc`.

```
Keys DefaultKeyBindings
{
    Shift<Key>Escape          window            f.post_wmenu
    Alt<Key>space             window|icon       f.post_wmenu
    Alt<Key>Tab               root|icon|window  f.next_key
    Alt Shift<Key>Tab         root|icon|window  f.prev_key
    Alt<Key>Escape            root|icon|window  f.next_key
    Alt Shift<Key>Escape      root|icon|window  f.prev.key
    Alt<Key>F6                window            f.next_key transient
    Alt Ctrl Shift<Key>exclam root|icon|window  f.set_behavior
}
```

You can modify or delete any of these bindings, except "`Alt Ctrl Shift<Key>exclam`", by editing or deleting the line. (In order for the editing to have an effect, the `keyBindings` resource in the `.Xdefaults` file must be set to `DefaultKeyBindings`.)

Modifying Keyboard Bindings and Their Functions

You can modify the keyboard bindings section of your `.mwmrc` 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]
}
```

The following keyboard binding contexts are recognized by the window manager:

root	Operates the function when the key is pressed while the root window has keyboard focus.
window	Operates the function when the key is pressed while a client window has keyboard focus.
icon	Operates the function when the key is pressed while an icon has keyboard focus.

Modifying Keyboard Bindings

Key bindings can be modified by:

- Editing the `DefaultKeyBindings` section in `.mwmrc`.
- Making a new key binding set.

To create a new keyboard binding set:

1. Edit `.mwmrc` to include the new key binding set with a unique name.
2. Set the `keyBindings` resource in your `.Xdefaults` file to the new name.

Switching Between Default and Custom Behavior

The window manager has a built-in key binding that allows you to switch back and forth between customized `mwm` behavior and default behavior. The key presses for doing this are Alt Shift Ctrl !.

The following client-specific resources are affected by this function:

`clientDecoration` `clientFunctions` `focusAutoRaise` `windowMenu`

Using the Window Manager with Multiple Screens

By default, the `mwm` manages one screen. Managing multiple screens can be specified in two ways:

- Using resources.
- Editing the startup command for `mwm`.

Using Resources to Manage Multiple Screens

The following resources configure the window manager to manage multiple screens:

- To specify that `mwm` manage multiple screens, use the resource:

  ```
  Mwm*multiScreen:   True
  ```

  This tells `mwm` to try to manage all screens that the server manages.

- To define the screen names, use the resource `screenList`. For example, the following resource names two screens `zero` and `one`.

  ```
  Mwm*screenList:  zero  one
  ```

Specifying Multiple Screens from the Command Line

You can use command-line options to start `mwm` so that it manages multiple screens.

- The `-display` option specifies the display. It has the syntax:

  `-display` *hostname*:*display*.*screen*

- The `-multiscreen` option causes `mwm` to manage all the screens on the specified display.

- The `-screens` option specifies the screen names used to obtain screen-specific resources.

For example,

```
mwm -display local:0.1 -multiscreen -screens zero one
```

causes `mwm` to manage all the screens on display 0. Screen 0 is named `zero`, and screen 1 is named `one`.

8

Using the X Clients

Programs running in the X environment can be divided into two groups:

X clients "Window-smart" programs written for the X Window System.

non-clients Programs written for terminals. Non-clients are run in terminal emulation windows.

Related chapter:

- Chapter 5 covers setting client, the display, and geometry resources.

Starting Clients and Non-clients

Programs can run as either background or foreground processes. In any X11 terminal window, you can run only one program as a foreground process, but you can run many programs 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 general syntax for the command line that starts a client is:

client [*-options*] [`&`]

An `&` at the end of the command line causes the client to start as a background process.

Programs can be started:

- From the command line.

  The client name and options are typed after the command line prompt.

- From menus.

  Refer to chapter 7 for details of how to create your own menus.

- As part of the X startup.

  Refer to chapter 4 for information about the `.x11start` file.

Command-Line Options

Command-line options override all default files. If no options are specified, the client is started using resource values from the resource database, the client's app-defaults, or from defaults built into the client.

Some **toolkit options** are common to most clients:

`-fn` *font*	Specifies the font to use for the client.
`-bg` *color*	Specifies the background color.
`-fg` *color*	Specifies the foreground color.
`-display` *host:display.screen*	Specifies the host where the client will display its output.
`-geometry` *width×height*	Specifies the size of the window and its location.
`-help`	Displays an explanation of the options available for the client.

For a specific client's options, refer to the client's man page.

Options have the syntax:

-option argument

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

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

Specifying the Display and Screen

The default display on which a client is displayed is obtained from the `DISPLAY` environment variable of the system on which the client starts. It sets the host, display number, and screen number to which the client directs its output. This is typically display 0, screen 0.

Most clients have a `-display` option that lets you set the host, display number, and screen on which the client will display its output. The `-display` option has the syntax:

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

host	The hostname of a valid system on the network.
display	The number of the display on the system on which you want the output to appear. A display can include more than one screen.
screen	The number of the screen where the output is to appear. The default is 0.

For example, executing the command:

```
hpterm  -display  hpxhere:0.1 &
```

starts an `hpterm` process on the local system and displays the window on display 0, screen 1 of the `hpxhere` system. The window has the default size, location, and color.

Starting Remote Programs

A remote client is a client that runs on a computer other than the computer running the X server. In other words, a remote client runs on one computer while its output is displayed on another.

There are several ways to run programs on a remote host from a command line:

- Use `rlogin` to log into the remote host.
- Use `remsh` to start a client remotely without formally logging in.

If the client produces output on a display, you must specify the display and screen on which you want the output to appear.

Running Programs Using 'rlogin'

You can use an existing terminal emulator window to log into a remote host. Once the window is acting as a terminal off the remote host, you can run clients there and direct the output to any display.

For example, the following commands log into and start `xload` on remote host `hpthere` and display the output on local system `hpxhere`.

```
rlogin  hpthere
xload  -display  hpxhere:0.0 &
```

Using 'remsh' to Start Programs

The benefit of using `remsh` instead of `rlogin` is that the the local system starts only one process (the client) with a remote shell; with the remote login, the local system starts both the remote login and the client.

Starting Clients Remotely

The following syntax starts a remote shell on a remote host, redirects `remsh` input, starts a client, and directs output to the local display.

`remsh` *remote* `-n` *client* `-display` *local*:*display*.*screen* `&`

remote	The remote host name.
client	Absolute path of the executable client file (`remsh` does not allow the `PATH` variable).
local	Local host name.

For example, the following command runs `xload` on remote host `hpthere` and directs output to the display of system `hpxhere`.

```
remsh hpthere -n /usr/bin/X11/xload -display hpxhere:0.0 &
```

Generally, `remsh` is preferred to `rlogin` for starting a remote program from a menu. For example, the following line added to the workspace menu starts a remote `hpterm` window on remote host `hpthere`:

```
"Doc files"  f.exec "remsh hpthere -n /usr/bin/X11/hpterm -display hpxhere:0.0 &"
```

Starting a Remote Non-Client

At the command-line prompt of an existing window, you could execute:

```
hpterm -display hpxhere:0.0 -e remsh hpthere -n ll &
```

This example starts a new `hpterm` client and directs its output to the local display (`-display hpxhere`). The `-e` option executes a remote shell on `hpthere` that connects the window to the remote host `hpthere` and lists the files in your home directory there. When the `ll` command finishes executing, the window created for it to run in will disappear. Thus, this method of starting remote non-clients is usually not desirable.

Stopping Programs

If a program has data you want to save, you must save the data *before* you stop it.

If a terminal window is running a non-client containing data, you must stop the non-client in the approved manner before you stop the window. Generally, a non-client has a "stop" provision, or stops when it has finished executing.

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 window 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.

If you are unable to stop a program in the normal manner, you should "kill" the program before you log out.

To kill a program, first try these keystrokes:

- Press CTRL c.
- Press CTRL d.
- Press q.
- Press ESC, then :, then q.

If these don't work, use the HP-UX `kill` command to stop the program's execution environment or "process." To use the `kill` command:

1. Save any data that needs saving.
2. Find the PID (process ID) by executing:

   `ps -fu` *login_name*

3. To kill the program, execute:

   `kill -2` *pid*

   where *pid* is the PID number. This is equivalent to CTRL c.
4. If this doesn't work, execute:

   `kill -3` *pid*

5. If this still doesn't work, execute:

   `kill -9` *pid*

Certain programs are *cached* during a session; that is, once they are started, closing them unmaps the window but does not stop the process. If you need to halt one of these processes during a session, use the `kill` command.

The X Clients

The following tables list the X clients described in this manual.

Initialization and Configuration Clients.

Client	Description	Covered in Chapter ...
`xmodmap`	Alters the modifier-key mappings of a keyboard.	9
`xset`	Adjusts display preference options for a session.	8
`xinitcolormap`	Initializes a new colormap for an X environment.	8
`rgb`	Creates a color database for X.	8
`xhost`	Adds a new remote host to your system.	8
`xrdb`	Loads a window manager's resource configuration into the server.	5
`xinit`	Starts the X server and selected clients.	4
`x11start`	Starts the X11 Window System using `xinit`.	4

Window Management Clients.

Client	Description	Covered in Chapter ...
`resize`	Sets the environment to reflect the correct window size.	8
`xwininfo`	Displays information about windows.	8

Graphics Functions Clients.

Client	Description	Covered in Chapter ...
`xseethru`	Opens a window into the graphics workstation image planes when the X Window System is running in the overlay planes.	11
`xwd`	Makes a pixmap screen dump in `xwd` format.	10
`xpr`	Prints a screen dump.	10
`gwind`	Creates a window for applications.	11
`gwindstop`	Stops multiple X windows.	11
`xwcreate`	Creates a new X window.	11
`xwdestroy`	Destroys an X window.	11
`xwud`	Displays a previously made screen dump.	10

Viewable Services Clients.

Client	Description	Covered in Chapter ...
`xterm`	Terminal emulator for a DEC or Tektronix terminal.	8
`hpterm`	Terminal emulator for HP Term0 terminals.	8
`dtterm`	EUC 4-byte capable DEC and Tektronix terminal emulator.	8
`xclock`	Displays an analog or digital clock.	8
`xload`	Displays the system load average.	8
`xsetroot`	Sets the color and appearance of the root window.	8

Font Management Clients

Client	Description	Covered in Chapter ...
`bdftopcf`	Compiles a BDF-formatted font into an X server format.	6
`mkfontdir`	Creates a `fonts.dir` file.	6
`xlsfonts`	Lists the fonts that match a given pattern.	6

The following clients do not require X to be running: `rgb`, `xpr`, `xwd2sb`, `sb2xwd`, and `mkfontdir`.

Clients Using Local Language

If you want a client to operate in a local language, be sure that the LANG environment variable is set and that the client's font is specified correctly.

Motif clients must have a font set specified in their `fontList` resource. `dtterm` and `hpterm` also require a `fontList` resource.

Terminal Emulation Clients

The X Window System has three terminal emulation clients, `hpterm`, `dtterm`, and `xterm`. The default for HP-UX is `hpterm`.

`dtterm` provides an EUC 4-byte capable terminal emulator. It emulates the DEC VT2200 terminal.

`hpterm` emulates an HP Term0 terminal.

`xterm` emulates DEC VT102 and Tektronix 4014 terminals.

To start a terminal emulator, type:

emulator `[-`*options*`] [&]`

There are too many options to cover here. Refer to the man page for the terminal emulators for all the options available.

The following example starts an `hpterm` emulator with scrollbars.

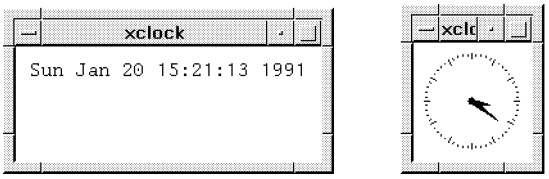
```
hpterm -sb
```

The following example starts a `dtterm` emulator with the title "my terminal" and display it initially as an icon:

```
dtterm -title "my terminal" -iconic
```

The 'xclock' Client

The `xclock` client displays an analog or digital clock. The digital clock also displays the day, date, time, and year; the format automatically varies for local language custom based on the value of the `LANG` environment variable.



Digital and Analog Clocks

The syntax for the `xclock` client is:

`xclock` [`-`*options*] [`&`]

For a complete list of `xclock` options, refer to the `xclock` man page.

The following example creates a digital clock that updates every 10 seconds.

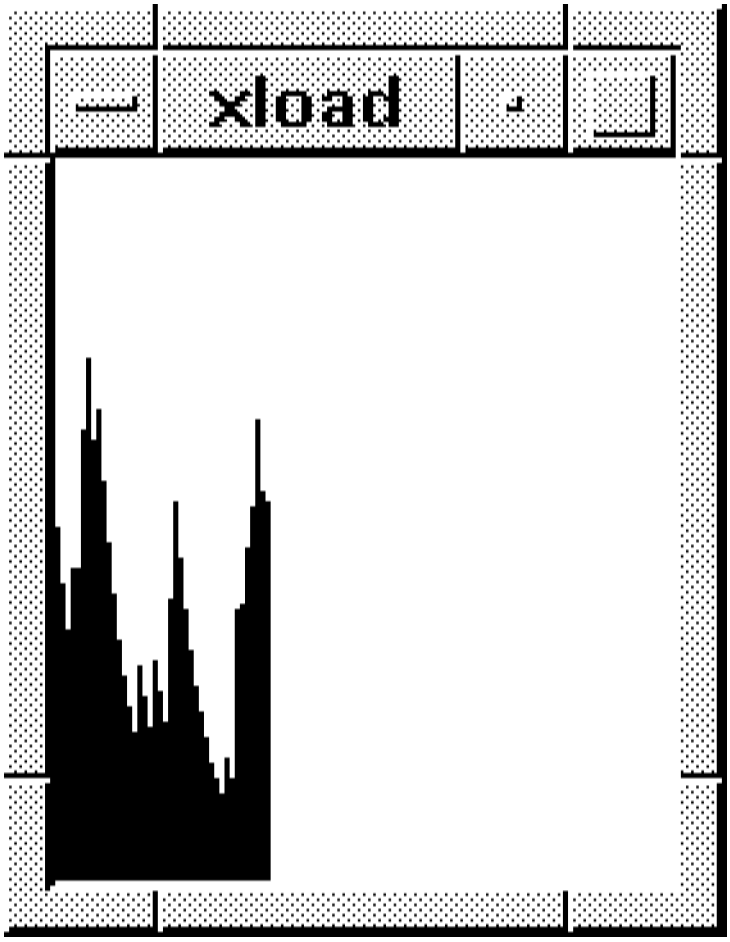
```
xclock -digital -update 10 &
```

The next example creates an analog clock that chimes every 30 minutes, updates every 5 seconds, and has yellow hands (all the other colors are the default ones).

```
xclock -analog -chime -update 5 -hd yellow &
```

The 'xload' Client

The `xload` client displays a periodically updated histogram of the system load.



The 'xload' Client

The syntax for the `xload` client is:

`xload` [`-`*options*]

where:

`-hl` *color*	The color of the scale lines.
`-jumpscroll` *number*	The number of pixels to shift the graph to the left when the graph reaches the right edge of the window. The default is half the width of the current window.
`-label` *string*	The string to put into the label above the histogram.
`-nolabel`	No label is displayed above the histogram.
`scale` *integer*	The number of tic marks in the histogram. The default is 1.
`update` *seconds*	The frequency at which the histogram is updated.

`xload` also accepts the toolkit command line options.

Customizing the Root Window with 'xsetroot'

The `xsetroot` client lets you:

- Customize the appearance of the root window.
- Change the bitmap used for the root window cursor.

The `xsetroot` client has the syntax:

```
xsetroot  [options]
```

where *options* are:

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

For example, the following command changes the workspace cursor using two custom bitmaps located in directory `$HOME/bits`.

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

Changing Display Preferences with 'xset'

The `xset` client allows you to change certain user preference options of the display. Note that hardware limitations and implementation differences may affect the results of the `xset` client.

`xset` provides a way to set:

- Bell volume, pitch, and duration.
- Keyboard click volume and autorepeat.
- Mouse acceleration and threshold.
- Font paths.
- Screen saver time.

The syntax for `xset` is:

`xset` *options*

where *options* are:

`-b`	Turns the bell off.
`b on/off`	Turns the bell on or off.
`b` *volume* `[,`*pitch*`, [,`*duration*`] ]`	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 in hertz and is specified together with a volume. *Duration* is in milliseconds and is specified with both volume and pitch. If only one parameter is given, it is taken as the volume. If two parameters are given, they are taken as volume and pitch.
`-c`	Turns the key click off.
`c on/off`	Turns the key click on or off.
`c` *0-100*	Specifies the key click volume as a percentage between 0 and 100.
`-fp/fp-` *path*`[,`*path...*`]`	Removes the specified directories from the font path.
`+fp/fp+` *path*`[,`*path...*`]`	Prefixes or appends the specified directories to the font path (depending on the position of the +).
`fp default`	Restores the default font path.
`fp` *path*`[,`*path...*`]`	Specifies the font path, absolutely.
`fp=` *path*	Sets the font path.
`fp rehash`	Causes the server to reread the `fonts.dir` file and the `fonts.alias`

	files in each path of the server's font path.
`m` *acceleration threshold*	Specifies the acceleration and threshold of the mouse. *Acceleration* indicates the change in mouse speed (for example: 2=double, 3=triple). *Threshold* indicates the number of pixels of movement required before acceleration takes place. If only one number is given, it is taken as the acceleration.
`m default`	Resets the mouse acceleration and threshold to their default values.
`p` *pixel color*	Controls color on a per pixel basis. *Pixel* is an integer representing a specific pixel in the X server's colormap. The exact number of pixels in the colormap depends on your hardware. *Color* specifies the color that pixel should be.
`pm default`	Restores the default font button codes to the pointer map.
`pm` *number*	Specifies the button codes for pointer map entries.
`-r`	Turns autorepeat off.
`r on/off`	Turns autorepeat on or off
`s` *length period*	Sets the screen saver option on. *Length* is the number of seconds that the server must be inactive before the screen is blanked. *Period* is the number of seconds a particular background pattern will be displayed before changing it.
`s blank`	Specifies that the screen saver should blank the video, if permitted by your hardware, rather than display the background pattern.
`s noblank`	Specifies that the screen saver should display the background pattern rather than blank the video.
`s expose`	Specifies that the server should discard window contents.
`s noexpose`	Specifies that the server should not enable the screen saver unless it saves window contents.

`s default`	Sets the system to its default screen saver characteristics.
`s on/off`	Sets the screen saver feature on or off.
`q`	Displays the current settings.
`-display` *host:display.screen*	Specifies the host, display number, and screen to be reset with `xset`.

Creating a Custom Color Database with 'rgb'

The `rgb.txt`, `rgb.pag`, and `rgb.dir` files in `/etc/X11` make up the color data base for the X Window System. It contains all the named colors and the amount of red, green, and blue needed to make the color. 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.

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 alternative spellings of the same color.

Depending on your needs, you may want to make your own custom color database modeled after the `rgb.txt` file.

Hewlett-Packard recommends that your custom color database have a name other than `rgb.txt`. You can either copy `rgb.txt` and make your changes, or start with an entirely new file. In either case, the file entries are in the following format:

redvalue greenvalue bluevalue name

The fields are separated by either tabs or spaces.

The `rgb.txt` file is the source file used by the `rgb` client to make two other files that are used by the server: `rgb.dir` and `rgb.pag`. If you run `rgb` without any parameters, it will use `rgb.txt`. If you want to use your custom database, use 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`.

To put your new color database into effect, you must add it to your `.x11start` file. For example, if your new database is composed of the files `2brite.txt`, `2brite.dir`, and `2brite.pag` in the `/home/ellen` directory, type the following command line to start your X environment:

```
.x11start -- -co /home/ellen/2brite
```

The server assumes the color database is in the `/etc/X11` directory unless told otherwise.

Note that recent X11 releases from Hewlett-Packard may contain more than one color database file, each customized for a particular display type for color consistency across display types. To avoid overwriting an existing `rgb.txt` file, the installation process for X11 does not automatically replace this file in `/etc/X11`, but installs the new `rgb.txt*` file(s) in the directory `/usr/newconfig` or `/etc/newconfig/X11R*`. You must manually process (using the `rgb` utility) the desired `rgb.txt*` file in order to use one of the new versions. You should copy the desired `rgb.txt*`, `rgb.dir`, and `rgb.pag` files to the `/etc/X11` directory. You may want to save the existing version of each file in `/etc/X11` before copying the new version in.

Initializing the Colormap with 'xinitcolormap'

The `xinitcolormap` client initializes 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 (for example, many applications that use Starbase graphics).

`xinitcolormap` has the following syntax:

```
xinitcolormap [options]
```

where the *options* are:

`-f` *colormapfile*	Specifies a file containing a colormap.
`-display` *display*	Specifies the server to connect to.
`-c` *count*	Only the first *count* colors from the colormap file will be used if this parameter is specified.
`-k` or `-kill`	Deallocate any colormap entries that were allocated by a previous run of `xinitcolormap`.

`xinitcolormap` choses a colormap file in the order shown below. Once one is found, then the other sources aren't searched.

1. The command line option [-f colormapfile].
2. `.Colormap` default value.
3. The `xcolormap` file in `/usr/lib/X11`.
4. If no colormap file is found, this default colormap specification is assumed— black (colormap entry 0), white, red yellow, green, cyan, blue, magenta (colormap entry 7).

`xinitcolormap` should be the first client program run at the start of a session in order to assure that colormap entries have the color associations specified in the colormap file. Sometimes you may encounter this X toolkit warning:

```
X Toolkit Warning: cannot allocate colormap entry for 94c4d0
```

where "94c4d0" is a color specified in the application running. If this occurs, it means that you have probably reached the limit of colors for your graphics card/display combination. Executing `xinitcolormap` may solve the problem.

For more information about `xinitcolormap`, refer to its man page.

Adding and Deleting Hosts with 'xhost'

Using `xhost`, you can add or delete a remote host's permission to access the local display server.

Note

Hosts entered by `xhost` have access only until the server recycles. A server recycles when the last client attached to a server goes away. For systems running many clients, this is usually at the end of a session. For systems running a server but no clients, hosts entered by `xhost` may be removed before you have a chance to use the remote host.

To add hosts permanently, make an entry in the `X0.hosts` file.

The `xhost` command is in the form:

```
xhost [+-] [name]
```

where:

`+`*name*	Add the remote host named *name* to the list of computers allowed to connect to the X server.
`-`*name*	Remove *name* from the list of computers allowed to connect to to the X server.
`+`	Allow access to everyone (access control disabled).
`-`	Allow access only to computers in the list (access control enabled).
blank	Display current status and list of computers allowed to access the X server.

For example, the following command allows the remote computer `hpgggggg` to access your local display.

```
xhost +hpgggggg
```

For more information, refer to the `xhost` man page.

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.

Use `resize` whenever you resize a terminal emulator 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.

The syntax for `resize` is as follows:

```
resize [options]
```

where *options* are:

`-c`	Resets the environment variables for `csh` shells.
`-h`	Uses Hewlett-Packard terminal escape sequences to determine new window size.
`-s` [*row col*]	Uses Sun escape sequences to determine new window size. New row and column sizes are specified with *row* and *col*.
`-u`	Resets the environment variables for `sh` and `ksh` shells.
`-x`	Uses VT102 escape sequences to determine new window size.

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 window 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 `.cshrc` 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`;}
```

Getting Window Information with 'xwininfo'

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

The syntax for `xwininfo` is as follows:

`xwininfo` *options*

where *options* are:

`-help`	Prints a summary of the command usage.
`-id` *id*	Specifies the target window by window id.
`-name` *name*	Specifies the target window by name.
`-root`	Specifies the root window as the target.
`-int`	Displays window information, normally shown as hexadecimal, as decimal.
`-tree`	Displays ids and names of the root, parent, and child windows.
`-stats`	Displays window id, location, size, depth, and other information as hexadecimal.
`-metric`	Displays height, width, x and y information in millimeters.
`-english`	Displays height, width, x and y information in inches, feet, yards.
`-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` *host:display.screen*	Specifies the host, display, and screen to target.

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 button 1.

```
xwininfo ==> Window id: 0x200013 (hpaaaaa)
  ==> Upper left X: 6
  ==> Upper left Y: 6
  ==> Width: 484
  ==> Height: 316
  ==> Depth: 8
  ==> Border width: 4
  ==> Window class: InputOutput
  ==> Colormap: 0x80065
  ==> Window Bit Gravity State: NorthWestGravity
  ==> Window Window Gravity State: NorthWestGravity
  ==> Window Backing Store State: NotUseful
  ==> Window Save Under State: no
  ==> Window Map State: IsViewable
  ==> Window Override Redirect State: no
  ==> Corners:  +6+6  -782+6  -782-694  +6-694
  -geometry =80x24+6+6
```

9

Customizing the Mouse and Keyboard

This chapter describes the following customizations:

- Changing mouse button actions.
- The `xmodmap` client.
- Going mouseless.
- Customizing keyboard input.

Related information:

- Chapter 7 contains `mwm` mouse and keyboard bindings.

Changing Mouse Button Actions

Normally, the mouse pointer buttons are mapped as follows:

Default Mouse Button Mapping.

Button Number	Button on a 2-button mouse	Button on a 3-button Mouse
Button 1	Left button	Left button
Button 2	Both buttons simultaneously	Middle button
Button 3	Right button	Right button
Button 4		Left and middle buttons simultaneously
Button 5		Middle and right buttons simultaneously

However, you can change these mappings. To generate buttons 4 and 5 on a three-button mouse, you must enable button chording as described later in this chapter.

Alternative Mouse Button Mappings.

To press Button	Left Hand Mapping		OSF/Motif Mapping	
	2-button mouse	3-button mouse	2-button mouse	3-button mouse
Button 1	Right button	Right button	Left button	Left button
Button 2	Both buttons simultaneously	Middle button	Right button	Middle button
Button 3	Left button	Left button	Both buttons simultaneously	Right button
Button 4		Middle and right buttons simultaneously		Left and middle buttons simultaneously
Button 5		Middle and left buttons simultaneously		Right and middle buttons simultaneously

The `xmodmap` utility can be used to change mouse button mappings. The syntax for changing mouse button mappings with `xmodmap` is:

```
xmodmap [ -e "[pointer = default                    ]" ]
        [                 pointer = number[ number...]]
        [ -pp                                         ]
```

`-e`	Specifies a remapping expression. Valid expressions are covered in "Customizing Keyboard Input" later in this chapter.
`default`	Set mouse keys back to default bindings
`number`	Specifies a list of button numbers to map the mouse keys to. The order of the numbers refers to the original button mapping.
`pp`	Print the current pointer mapping.

For example, to reverse the positions of buttons 1 and 3 for left-handed mapping:

```
xmodmap -e "pointer = 3 2 1"       2-button mouse
xmodmap -e "pointer = 3 2 1 5 4"   3-button mouse
```

To establish OSF/Motif-standard button mapping:

```
xmodmap -e "pointer = 1 3 2"       2-button mouse
xmodmap -e "pointer = 1 3 2 4 5"   3-button mouse
```

`xmodmap` is discussed in more detail in "Modifying Modifier Key Bindings with 'xmodmap'".

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 `X0pointerkeys` in `/usr/lib/X11`.

The `X*pointerkeys` file lets you specify:

- The keys that move the pointer.
- The keys that act as pointer 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 screens.
- That button chording is enabled or disabled.
- That button latching is enabled or disabled.
- Tablet subsetting.
- Screen switching behavior for multi-screen configurations.

If you modify a `X*pointerkeys` file, it does not take effect until you restart the X Window System again.

Configuring 'X*devices' for Mouseless Operation

If you have only one keyboard and no pointer device, 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 or more 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 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 `XOpointerkeys` file only if one or more of the following statements are true:

- You want to use the keyboard for a pointer.
- You want to change the pointer keys from their default configuration.
- You use the `XOscreens` 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 pointer.
- You want to change the pointer keys from their default configuration.
- You use a configuration file other than the `XOscreens` file to configure your display.

Syntax

You assign a keyboard key to a mouse function (pointer movement or button operation) by inserting a line in the `X*pointerkeys` file. Lines in the `X*pointerkeys` file have the 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, make sure that the key you are assigning doesn't already serve a vital function.

You can assign keyboard keys to pointer directions by specifying options in an `X*pointerkeys` file. The following table lists the pointer movement options, the `X*pointerkeys` functions that control them, and their default values:

Pointer Movement Functions.

Movement Option	Function	Default Key
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:

Pointer Distance Functions.

Movement	Function	Default
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:

Button Operation Functions.

Button Operation	Function	Default Key
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 mapping of buttons on the pointer by using options in the `X*pointerkeys` file. The following table lists the `X*pointerkeys` functions that control button mapping and their default values. Like `xmodmap` and `xset`, these functions affect only the X pointer, not any extension input devices.

Button Mapping Functions.

Button Mapping	Function	Default Key
Set button 1 value	`button_1_value`	1
Set button 2 value	`button_2_value`	2
Set button 3 value	`button_3_value`	3
Set button 4 value	`button_4_value`	4
Set button 5 value	`button_5_value`	5

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:

Reset and Threshold Functions.

Option	Function	Default Key
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 screens.	`screen_change_amt`	30 pixels 0 if a graphics tablet is used

`screen_change_amt` is used only if your system is configured for more than one screen. (Refer to "Using Custom Screen Configurations" in Chapter 3). `screen_change_amt` enables you to avoid switching from one screen to another if you accidentally run the pointer off the edge of the screen. `screen_change_amt` establishes a "distance threshold" that the pointer must exceed before the server switches screens. As the previous table shows, the default width of the threshold is 30 pixels, but acceptable values range from 0 to 255.

When a graphics tablet is used as the X pointer, the `screen_change_amt` defines an area at the left and right edges of the tablet surface that will be used to control screen changes. Moving the puck or stylus into the left or right area will cause the X server to switch to the previous or next screen.

Button Chording

Option	Function	Default Action
Turn button chording off or on.	`button_chording`	ON for devices with 2 buttons, OFF for devices with >2 buttons.

Button chording refers to the generation of a button by pressing two other buttons. If you have a two-button mouse, you can generate button 3 by pressing both buttons together. With a three-button mouse, you can generate button 4 by pressing the left and middle buttons together and button 5 by pressing the middle and right buttons together. See the button chording examples in the `X*pointerkeys` file.

You can also use the `X*pointerkeys` file to configure pointer buttons so they are latched. When this feature is enabled, a button you press stays logically down until you press it again. See the example `X*pointerkeys` file in `/usr/lib/X11` for information on configuring this functionality.

Note

The sample `X*pointerkeys` file is placed in `/usr/lib/X11` at install time. If you subsequently update your system, the `X*pointerkeys` file in `/usr/lib/X11` is *not* overwritten, and the sample file is placed in `/usr/newconfig`.

Specifying a Portion of a Tablet

Option	Function	Default

Use a subset of the tablet surface as the X pointer device	`tablet_subset_width`
	`tablet_subset_height`
	`tablet_subset_xorigin`
	`tablet_subset_yorigin`
		disabled

If a tablet is used as the X pointer device, it may be desirable to use only a portion of the tablet surface. A rectangular subset of the surface may be specified with these functions. The units are in millimeters from the upper left corner of the tablet surface. For example, if you want to use only an "A" size portion of a larger "B" size tablet, the following lines could be added to the `X*pointerkeys` file:

```
tablet_subset_xorigin   68
tablet_subset_yorigin   40
tablet_subset_width     296
tablet_subset_height    216
```

You can also use the `X*pointerkeys` file to control screen switching behavior in multi-screen configurations. See the example `X*pointerkeys` file in `/usr/lib/X11` for an example of this functionality.

Note

The sample `X*pointerkeys` file is placed in `/usr/lib/X11` at install time. If you subsequently update your system, the `X*pointerkeys` file in `/usr/lib/X11` is *not* overwritten, and the sample file is placed in `/usr/newconfig`.

Modifier Keys

You can select up to three keys from among the two Shift keys, the two Extend char keys, and the CTRL key and use them each as **modifier keys**. 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 *hpterm cursor* to the right, you press ▶ as usual. To move the *x server pointer* to the right, you press Left Shift ▶.

Specifying Pointer Keys

To find out what key names are valid for the keyboard you are using, enter

```
xmodmap -pk
```

You may also use the *default* X Keysymbol names assigned to these keys by the X Server.

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 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 `XOpointerkeys` 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 only the Left Shift key (and not the Right Shift) modifies the press of an arrow key from cursor to pointer movement.

Now, 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. Furthermore, another application 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 Left Shift ▲ moves the *cursor* up in the program you frequently operate.
- Pressing Left Shift Left Extend char ▲ moves the *pointer* up.

Using a similar technique, you can also reassign the CTRL Left Shift Reset sequence that aborts a session. You can specify the press of a single key 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.

Note

There are now two keyboards available for Hewlett-Packard workstations, the 46021 keyboard, and the C1429 keyboard. See appendix B, Using the Keyboards, for more information on using these keyboards and the differences between them.

The syntax for `xmodmap` is as follows:

`xmodmap` *options* [*filename*]

where *options* are:

`-display` *host: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` *expression*	Specifies a remapping expression to be executed.
`-pm, -p`	Prints the current modifier map to the standard output. This is the default.
`-pk`	Prints the current keymap table to the standard output.
`-pp`	Print the current pointer map to the standard output.
`-`	Specifies that the standard input should be used for the 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:

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 that uniquely identifies each key on a keyboard. Values may be in decimal, octal, or hexadecimal.

`keysym` Refers to the character symbol name associated with a keycode, for example, KP_Add.

`modifier` Specifies one of the eight modifier names.

The following are the modifier names available for use in keyboard customization:

Valid Modifier Names.

Modifier Names			
Shift	Control	Mod2	Mod4
Lock	Mod1	Mod3	Mod5

On Hewlett-Packard keyboards, the `lock` modifier is set to the Caps key. However, any of the modifiers can be associated with any valid key symbol. Additionally, you can associate more than one key symbol with a modifier (such as Lock = Shift_R and Shift_L), and you can associate more than one modifier with a key symbol (for example, Control = Caps_Lock and Lock = Caps_Lock).

For example, on a PC-style keyboard, you can press d to print a lower case "d", Shift d to print a capital "D", Alt d to print something else, and Shift Alt 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.

Examples

Suppose you frequently press 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. 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
```

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 88 = F1
keycode 55 = 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
```

On a larger scale, you can change your current keyboard to a Dvorak keyboard by creating a file with the appropriate keyboard mappings.

```
xmodmap .keymap
```

Printing a Key Map

The -pk option prints a list of the key mappings for the current keyboard.

```
xmodmap -pk
```

The list contains the keycode and up to four 2-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. Each column is in two parts: hexadecimal key symbol value, and key symbol name.

10

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.

SharedPrint/UX is available on HP-UX 10.0 systems. If you want to use it instead of the printing techniques described in this chapter, refer to *SharedPrint/UX User and Administrator's Guide*.

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 are `xwd` screen dumps.

The syntax for `xwd` is as follows:

`xwd [`*options*`]`

where *options* are:

`-help`	Provides a brief description of usage and syntax.
`-id` *id*	Specifies the window to be dumped by its *id* rather than using the mouse to select it.
`-add`	Adds value to every pixel.
`-name` *name*	Specifies the window to be dumped by its *name* rather than using the mouse to select it.
`-root`	Specifies that the window to be dumped is the root window.
`-add` *value*	Add *value* to every pixel. *value* is signed.
`-nobdrs`	Dumps the window without borders.
`-out` *filename*	Specifies that the screen dump is to be stored in the file *filename*.
`>` *filename*	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` *display*	Specifies the screen that contains the window to be dumped.

This first example stores a window in a file named `savewindow`, using the pointer to determine which window you want.

1. Display 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 any pointer button. After the image is captured, the cursor changes back to its normal shape and the window is stored in the file `savewindow`.

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 `xwud` client allows you to display an `xwd`-format 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 display on which it is to be restored. For example, an image created and stored using a depth of four cannot be restored on a display with a different depth.

The syntax for `xwud` is as follows:

```
xwud [options]
```

where *options* are:

`-help`	Displays a brief description of the options.
`-in` *filename*	Specifies the file containing the screen dump.
`-inverse`	Reverses black and white from the original monochrome dump.
`-display` *host:display.screen*	Specifies the screen on which to display the dump.

This example displays the xwd-format 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.

Refer to the system administrator manual(s) for your system if you need to:

- 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`.

`xpr` [*options*] *filename*

where *options* are:

`-scale` *scale*	Specifies a multiplier for pixel expansion. The default is the largest that will allow the entire image to fit on the page.
`-density` *dpi*	Specifies the dots per inch for the printer.
`-height` *inches*	Specifies the maximum height in inches of the window on the page.
`-width` *width*	Specifies the maximum width in inches of the window on the page.
`-left` *inches*	Specifies the left margin in inches. The default is centered.
`-top` *inches*	Specifies the top margin in inches. The default is centered.
`-header` *caption*	Specifies a caption to print above the window.
`-trailer` *caption*	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` *filename*	Specifies a file to store the output in.
`-append` *filename*	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` *n*	Prints the window on *n* pages. Not applicable to HP printers.
`-device` *dev*	Specifies the printer to use.

`ljet`	HP LaserJet series, HP ThinkJet, HP QuietJet, RuggedWriter, HP2560 series, HP2930 series, other PCL devices.
`pjet`	HP PaintJet (color mode).
`pjetxl`	HP PaintJet XL.
`ln03`	DEC LN03.
`la100`	DEC LA100.
`ps`	PostScript printers.
`pp`	IBM PP3812.

`-cutoff` *level*	Specifies intensity for converting color to monochrome for printing on a HP LaserJet printer.
`-noposition`	Bypasses header positioning, trailer positioning, and image positioning commands for the HP LaserJet and HP PaintJet printers.
filename	Specifies the `xwd` file to print.

For example, suppose you want to print a `xwd` file named `myfile` that you previously created with `xwd`. You want to print the file on a HP LaserJet printer in portrait mode with black and white the reverse of the original `xwd` file.

```
xpr -device ljet -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 image is translated into a grid of the size you specify. For example, if you specify a scale of 5, each bit in the image 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 necessary to 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

Use the device name `pjet` to direct output to a HP PaintJet printer.

For example, the following command prints a `xwd` file named `myfile` on a HP PaintJet printer.

```
xpr -device pjet myfile Return
```

Color images printed on a HP LaserJet printer will be in black and white instead of color.

`xpr` prints only in black and white, no shades of gray. If your original color image contained many colors of the same intensity, the HP LaserJet printer 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.

If you want color images to print in shades of gray on your LaserJet, use the Starbase utility `pcltrans` instead of `xpr`. Refer to the Starbase documentation for information.

11

Using Graphics With X Windows

This chapter covers the following topics:

- Window-smart and window-naive applications.
- Opening and destroying windows.
- Creating transparent windows.

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 X Window System.

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 program you want to run.

For example, the following command will start a hypothetical Starbase application named `planetarium` that displays a moving view of the night sky. Assume that the program is in the `/home/ellen/funstuff` directory on your computer.

```
/home/ellen/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). Some old 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 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 and stop it for you.*

The next sections cover:

- Creating a window
- Destroying a window

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.

Use `xwcreate` from the command line.

```
xwcreate [options]
```

where:

`-display` *host:display.screen*	Specifies the screen the window will appear on
`-parent` *parent*	Names a window to be the parent of the window being created.

`-geometry` *width*×*height*±*col*±*row*	Specifies desired size and location of window.
`-r`	Specifies backing store. Default is no backing store.
`-bg` *color*	Specifies the background color. The default is black.
`-bw` *pixels*	Specifies the border width in pixels. The default is 3 pixels wide.
`-bd` *color*	Specifies the border color. The default is white.
`-depth` *depth*	Specifies the depth of the window. The default is the same depth as its parent.
`-visual` *visualclass*	Specifies the visual class of the window when multiple visual classes are supported by the display at the specified depth.
`-overlay`	Specifies that an overlay plane visual should be used.
`-wmdir` *directory*	Specifies the name of the directory containing the pty file for the window.
`-title` *name*	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 is created with the same depth as its parent, or with the same depth as the root if no parent is specified. 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 terminate.

Use `xwdestroy` from the command line.

`xwdestroy` [`-wmdir` *path/directory*] *window1 window2 ...*

where:

`-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 terminate `gwind`. 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.

Use `gwindstop` from the command line.

`gwindstop` [*directory*] [*directory*] ...

directory — The directory containing the pty files for the windows to be destroyed.

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.

Use `xseethru` from the command line.

`xseethru` [`-geometry` *width*×*height*±*col*±*row*] [`-display` *host:display.screen*]

where:

`-geometry` — The geometry used to create the window.

`-display` — The screen the window will appear on.

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 to make the root window transparent when you are running X in the overlay planes.

Use `xsetroot` from the command line.

`xsetroot` [`-solid` *color*]

where:

`-solid` Sets the window color to *color*.

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.

This example opens an `hpterm` window with a transparent background color.

```
hpterm -bg transparent Return
```

A

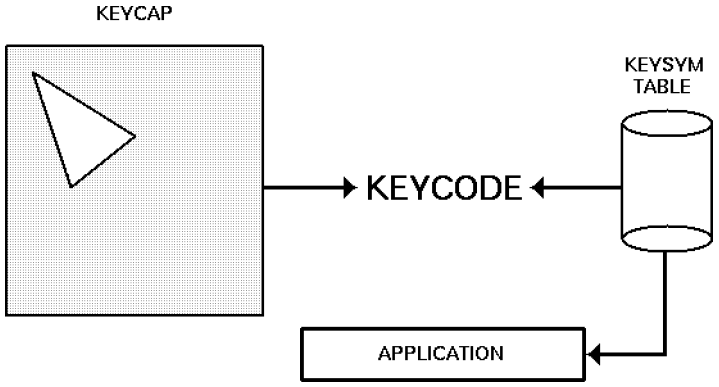
Using the Keyboards

There are now two keyboards available for Hewlett-Packard workstations. In addition to the 46021 keyboard, a personal computer-style keyboard, C1429 is also available. This new keyboard is also known as the "Enhanced Vectra" keyboard.

Understanding the Keyboards

If an application is reading input directly from the keyboard, it receives a *keycode* when a key is pressed. Equivalent keys on the two keyboards are those that generate the same keycode. If an equivalent key does not exist, there is no way to generate the corresponding keycode.

In an X Window System environment, keycodes are mapped into *key symbols* by the X library. The key symbols are stored in a *keysym table*. Application programs then reference these key symbols when accessing keys.



Keycap, Keycode, and Keysym Relationships

Equivalent keys are those keys that are mapped to the same key symbol. One advantage of this mapping is that if a key does not physically exist on a keyboard, its equivalent key symbol can be mapped to some other key through the corresponding keycode.

Default Keyboard Mapping

The default keyboard mapping supplied with the X Window environment maps the C1429 keyboard to the same key symbols that are used for the 46021 keyboard. This allows existing X client programs that expect to receive input from a 46021 keyboard to be used with either keyboard. However, the result is that some keys on the C1429 keyboard are mapped to key symbols that do not match the engravings on their keycaps.

Equivalent Keys

Some applications may expect to use keys that exist on one of the keyboards but not the other. In most cases, if a key does not exist on the keyboard in use, it is still possible to use some other key that is equivalent. To do this, it is necessary to know which keys are equivalent on the two keyboards.

There are 14 keys on the C1429 keyboard that generate keycodes equivalent to keys on the 46021 keyboard, but have different engravings on the keycaps. Some have the same key symbol on both keyboards, while others do not. These C1429 keys, their 46021 equivalents, and the corresponding symbol names are shown in the following table.

C1429 Keycap	46021 Keycap	Default Key Symbol	XPCmodmap Symbol
F9	blank1	F9	F9
F10	blank2	F10	F10
F11	blank3	F11	F11
F12	blank4	F12	F12
PrintScreen/sysRq	Menu	Menu	Print
Scroll Lock	Stop	Cancel	Scroll_Lock
Pause/Break	Break/Reset	Break/Reset	Pause/Break
Page Up	Prev	Prior	Prior
Num Lock	System/User	System/User	Num_Lock
End	Select	Select	End
Page Down	Next	Next	Next
Enter	Return	Return	Return
Alt (left)	Extend char (left)	Meta_L	Alt_L
Alt (right)	Extend char (right)	Meta_R	Alt_R

Changing Key Mapping

X provides the means to change the key mapping, if you so desire. One way to accomplish this is by running the `xmodmap` client program. Hewlett-Packard provides two files in the directory `/usr/lib/X11` to use with `xmodmap`. One, `XPCmodmap`, causes `xmodmap` to change the key mapping to match the keycap engravings on the C1429 keyboard. The other, `XHPmodmap`, causes `xmodmap` to change the key mapping to match the keycap engravings on the 46021 keyboard, which are the defaults. This allows either keyboard to be used with applications that expect the other keyboard, although only one mapping can be used at any given time. When the mapping is changed, the X Server notifies all clients that are executing at that time. Some clients may load the new mapping from the server right away, but others may have to be restarted in order to recognize the new mapping. For more information about using the `xmodmap` client, see the `xmodmap` man page. Additional information can be found in Chapter 9.

C1429 Keyboard

Execute the following command to change the mapping of the keys shown above to match the engravings on the C1429 keycaps.

```
/usr/bin/X11/xmodmap /usr/lib/X11/XPCmodmap
```

46021 Keyboard

Execute the following command to change the mapping to match the 46021 keyboard.

```
/usr/bin/X11/xmodmap /usr/lib/X11/XHPmodmap
```

Comparing the Keyboards

The 46021 keyboard has 107 keys, while the C1429 keyboard has 101 keys. There are 7 keys on the 46021 keyboard whose keycodes cannot be generated by any key on the C1429 keyboard, and whose key symbols cannot be generated when using the default keymap for the C1429 keyboard. The missing keys are:

- Clear line
- Clear display
- Insert line
- Delete line
- Print/Enter
- . (on number pad)
- Tab (on number pad)

. and Tab exist elsewhere on the C1429 keyboard, and the others are not needed by most applications. Applications that do need one or more of them must assign their key symbols to the keycodes of existing keys. The `xmodmap` client can be used to determine the keycode-to-key symbol mapping of existing keys, and it can also be used to assign the key symbol to the desired keycode. These keys use

HP specific key symbol names whose correct spelling can be found in the file `/usr/lib/X11/XKeysymDB`.

The Right Control key on the C1429 keyboard generates a keycode that has no equivalent on the 46021 keyboard. This key has the same effect as the Left Control key by default.

Keys not mentioned above exist on both keyboards, and have the same key symbols.

Glossary

Accelerator
A key or sequence of keys (typically a modifier key and some other key) that provides a "shortcut," for accessing functionality.

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.

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.

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.

bitmap font
A bitmap font is made from a matrix of dots.

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.

button mapping
Association of a button number with a physical mouse button.

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.

cluster
A network of computers in which only one computer has file-system disk drives attached to it.

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. Every time 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 by the X Window System, 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 `/home/`*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.

input device
Any of several pieces of equipment used to give information to the system. Examples are the keyboard, a mouse, or a digitizer tablet.

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.

mouseless operation
Although a mouse makes it easy to use the X Window System, the mouse is not absolutely necessary. The system can be configured to run from the keyboard alone.

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.

parent window
A window that causes another window to appear. A window that "owns" other windows.

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 are usually interchangeable.

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.

scalable fonts
Scalable fonts are defined by a file containing a mathematical outline used by the system to create a bitmapped font for a particular size, slant, or weight.

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 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.

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* or *Terminal Control: User's Guide*.

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. Several terminal emulators are supplied with X11—`hpterm`, which emulates HP terminals, `xterm`, which emulates DEC and Tektronix terminals, and `dtterm`, which emulates a DEC VT2200 terminal and has EUC 4-byte capability.

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 a 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.