X-WINDOWS

A Graphical User Interface for the
Coherent Operating System.

Mark Williams
Company

Revision 2 Printing 5 4 3 2 1

Published by Mark Williams Company, 60 Revere Drive, Northbrook, Illinois 60062.

Sales: Phone: (800) MARK-WMS
FAX: (708) 291-6750
E-mail: sales@mwc.com
Technical Support:
Phone: (708) 291-6700
FAX: (708) 291-6750
E-mail: support@mwc.com
CompuServ: 76256,427

This manual was written under the COHERENT operating system, using the MicroEMACS text editor. Text formatting was performed by the COHERENT edition of the **troff** text formatter, using its PostScript output function. Capitals and ornaments are derived from the Golden Bible of Augsburg, and were supplied in encapsulated PostScript form by BBL Typographic, 137 Narrow Neck Rd., Katoomba, NSW 2780, Australia. The key-caps font was supplied by SoftMaker, Inc., 2195 Faraday Avenue, Suite A, Carlsbad, CA 92008. Page design was implemented with custom-written macros written in the **troff** text-formatting language and in PostScript. Typesetting of this manual, from the table of contents through the index, was performed by one script written in the COHERENT Bourne shell. Camera-ready copy was printed on a Hewlett-Packard LaserJet IIP printer using the Pacific Page PostScript cartridge.

Table of Contents

Introduction to COHERENT X 1
How to Use This Manual 1
System Requirements 1
Contents of This Package 1
Bibliography 2
Technical Support 2
Help Us Help You 3
Where To Go From Here 3
Using the X Window System 5
Introduction to a Graphical Interface 5
Using the Mouse 6
Moving and Resizing Windows 6
Screen Buttons 8
Sliders 9
Menus 9
Introduction to xvt 14
Cutting and Pasting 15
Introduction to xclock 16
The Applications Menu 17
Shutting Down X 18
Conclusion 18
A Tour of the X System 18
The Structure of X 19
Background 19
X Architecture 20
Libraries, Widgets, and Resources 21
Bit Maps 22
Fonts 22
Colors 22
Customing the Window Manager 23
Customizing xinitrc 23
Customizing .twmrc 25
Variables 25
Bindings 28
Menus 29
Where To Go From Here 30
X Windows Clients 31
X Utilities 31
Bit Maps 31
Colors 32
Fonts 32
Manipulating the Console 33
Programming Tools 33
Resources and Properties 34
System Monitoring 35
Miscellaneous Utilities 36
Clients 37
Games 37
Observing the System 38
Pretty Pictures 38
Timepieces 38
Tools 38
Customizing X Programs 39
Resources 39
Modifying Applications 40
Modifying a Font Resource 41

Where To Go From Here . . . 42
Recompiling X Applications . . . 43
Imakefiles and Makefiles . . . 43
Modifications to Makefiles . . . 43
Problems Seen During Compilation . . . 44
Recompiling an Example Application . . . 45
Building Your Own Makefile . . . 46
Where To Get X Sources . . . 47
The Lexicon . . . 49
appres . . . List an application´s resource data base . . . 51
atobm . . . Convert ASCII to an X bit-mapped image . . . 51
bdftopcf . . . Generate a PCF font from a BDF file . . . 52
bitmap . . . Bit map editor . . . 53
bmtoa . . . Convert an X bit-mapped image to ASCII . . . 60
editres . . . Resource editor for X Toolkit applications . . . 60
ico . . . Animate an icosahedron or other polyhedron . . . 66
imake . . . C preprocessor interface to the make utility . . . 67
listres . . . List resources in widgets . . . 70
maze . . . Create and solve a random maze . . . 70
mkdirhier . . . Make a directory hierarchy . . . 71
mkfontdir . . . Create file fonts.dir from directory of font files . . . 71
oclock . . . Display an analogue clock . . . 72
puzzle . . . The X scrambled-number game . . . 73
resize . . . Set environmental variables to show window size . . . 74
showrgb . . . Un-compile an RGB color-name data base . . . 74
startx . . . Initiate an X session . . . 74
twm . . . Tab Window Manager for the X Window System . . . 75
viewres . . . Graphical class browser for Xt . . . 92
X . . . X Window System server . . . 95
X clients . . . 98
X utilities . . . 98
x11perf . . . Test performance of the X11 server . . . 100
x11perfcomp . . . Compare the output of multiple runs of x11perf . . . 105
xauth . . . Display/edit authorization information . . . 106
xbiff . . . Notify the user that mail has arrived . . . 107
xcalc . . . Scientific calculator for X . . . 110
xclipboard . . . Hold multiple selections for later retrieval . . . 115
xclock . . . Display a clock . . . 117
xcmsdb . . . Manipulate xlib screen-color characterization data . . . 119
xcmstest . . . XCMS test program . . . 120
xcutsel . . . Copy text between the cut buffer and the primary selection . . . 121
xdpyinfo . . . Display information about an X server . . . 123
xedit . . . Simple text editor for X . . . 123
xev . . . Print contents of X events . . . 126
xeyes . . . Display two roving eyes . . . 126
xfd . . . Display all the characters in an X font . . . 127
xfontsel . . . Interactively select X11 fonts . . . 129
xgas . . . Animated simulation of an ideal gas . . . 132
xgc . . . X graphics demonstration . . . 133
xinit . . . Initiate the X Window System . . . 134
xkill . . . Kill an X client . . . 136
xload . . . Display your system´s load average . . . 136
xlogo . . . Display the X Window System logo . . . 137
xlsatoms . . . List atoms defined on server . . . 138
xlsclients . . . List client applications running on a display . . . 139
xlsfonts . . . List fonts being used on a server . . . 140
xmag . . . Magnify a part of the screen . . . 140
xmkmf . . . Control the building of a Makefile . . . 142
xmodmap . . . Modify X keymaps . . . 142
xpr . . . Print a dump of an X window . . . 145
xprop . . . Display an application´s properties . . . 149

xrdb . . . . . Read/set the X server´s resource data base 152
xrefresh. . . . . Refresh all or part of an X screen. 155
xset . . . . . Set preferences for the display 156
xsetroot. . . . . Set preferences for the root window 158
xstdcmap. . . . . X standard color-map utility 159
xterm . . . . . Terminal emulator for X 160
xtetris. . . . . Wildly amusing implementation of Tetris 177
xvt. . . . . VT100 emulator. 178
xwd . . . . . Dump an image of an X window 181
xwininfo . . . . . Display information about a window. 182
xwud . . . . . Un-dump a window image. 183
Index 185

Introduction to COHERENT X

Congratulations on your purchase of X Windows for COHERENT! This product is a port of XFree386 release 1.2, which implements release 11, revision 5, of the Massachusetts Institute of Technology's X Window System. The X Window System (or "X Windows" for short) lets you use your keyboard and mouse to open multiple windows and run multiple graphics programs simultaneously on your screen. With X Windows, you can use on your COHERENT system the wealth of graphics-oriented programs — games, graphics-manipulation programs, images, tools, and utilities — available at minimal cost from many public sources.

How to Use This Manual

This manual consists of the following:

- An introduction — what you are reading right now.
- An introduction to the X Window System and the window manager **twm**, including directions on how to customize X Windows and **twm** to suit your tastes.
- An introduction to the X utilities and clients, and directions on how to use resources to modify clients.
- A discussion of how to recompile X programs. This covers such topics as how to construct a **Makefile** from an **Imakefile**, and commonly encountered problems.
- A Lexicon of manual pages for every X utility and client included with X Windows for COHERENT. Note that every Lexicon entry is also available on-line, and can be read by using the COHERENT command **man**.

If you are experienced in using X, you can jump to the Lexicon for a summary of the utility or client that interests you. The Lexicon entries **X clients** and **X utilities** list the clients and utilities included with X Windows for COHERENT. If you are a beginner, however, we suggest that you read through the tutorials at the beginning of this manual. They will help you understand how X works, and so help you to become productive more quickly.

System Requirements

To run this X Windows for COHERENT, you need the following:

- A computer that is running COHERENT release 4.2 or later.
- At least four megabytes of RAM. At least eight megabytes is needed to run the color X server; 16 megabytes is preferred.
- Twenty-five megabytes of hard-disk space on the file system that holds directory **/usr/X11**. More is preferred.
- A serial mouse — that is, a mouse that is plugged into a serial port on your computer. Note that X Windows does *not* work with a bus mouse — that is, a mouse that is plugged directly into your system's mother board. We recommend a three-button mouse, although two-button mice can also be used. This implementation of X works with mice from Microsoft, Mouse Systems, and Logitech (Mouseman), and with devices that mimic one of the above, such as those manufactured by Honeywell.
- A VGA or SVGA video board and monitor.

Contents of This Package

X Windows for COHERENT includes all of the tools you need to run X on your system. It also includes tools with which you can import and write X applications. The package includes the following:

- A large selection of X clients and utilities.
- Object modules from which the included X programs are linked.
- Bit-mapped images that can be displayed by an X client.
- Sample and default configuration files for X programs.
- A selection of fonts.

- Header files used by X clients and utilities.
- Libraries of X functions.

Please note that the included libraries do not support a mathematics co-processor.

Bibliography

The X Window System is a huge and elaborate system, for which a huge literature exists. This manual can only scratch the surface of X. It walks you through installing, invoking, and configuring X Windows, and introduces the tools and clients. However, much of X lies beyond the scope of this manual, especially if you wish to import or write X applications. We recommend the following books for further reading.

If you are an experienced programmer and you wish to import or write your own X Windows applications, there is no substitution for the first five volumes of *The X Window System,* published by O'Reilly & Associates, Inc. Contact your local bookseller, or write to O'Reilly & Associates, Inc., 632 Petaluma Avenue, Sebastopol, CA 95472 (1-800-338-6887). Another useful volume is *The X Window System in a Nutshell,* second edition, edited by Ellie Cutler, Daniel Gill, and Tim O'Reilly. (Sebastopol, Calif., O'Reilly & Associates, Inc., 1992), which is a handy pocket reference for people who already know X internals.

If you are new to the X Window System, we recommend the following two books:

- Mansfield, Niall: *The Joy of X: An Overview of the X Window System.* New York, Addison-Wesley Publishing Co., 1993.
- Quercia, Valerie, O'Reilly, Tim: *The X Window System.* Vol. 3, *X Window System User's Guide,* ed 4. Sebastopol, Calif., O'Reilly & Associates, Inc., 1992.

Technical Support

Mark Williams Company provides free technical support to all registered users of X Windows for COHERENT. If you are experiencing difficulties with this product, feel free to contact the Mark Williams Technical Support Staff. You can telephone, send electronic mail, or write. Please note that this support is available *only* if you have returned your User Registration Card for COHERENT.

Before you contact Mark Williams Technical Support with your problem, *please check this manual first.* If you do not find an article in the Lexicon that addresses your problem, be sure to check the index at the back of the manual. Often, the information that you want is kept in an article that you didn't consider, and the index will point you to it.

Another good way to find a topic in the manual is to use the command **apropos**, which is part of the COHERENT system. **apropos** finds every article in the Lexicon that mentions a given word or phrase. For details on how to use this command, see its entry in the COHERENT Lexicon.

If the manual does not solve your problem — or if you find it to be misleading or difficult to understand — then Mark Williams Technical Support is available to help you. You can reach Technical Support via the following routes:

Electronic Mail
If you have access to the Internet, send mail to **support@mwc.com**. This is the preferred means of communication. Be sure to include your surface address and telephone number as well as your e-mail address, so we can contact you even if return electronic mail fails.

FAX Send your technical FAXes to 1-708-291-6750.

Surface Mail
Write to Technical Support, Mark Williams Company, 60 Revere Drive, Northbrook, IL 60062.

Telephone
To contact Technical Support via telephone, call 1-708-291-6700, between 9 AM and 5 PM, Central Time. Please have your manual at hand. Please collect as much information as you can concerning your difficulty before you call. If possible, call while you are at your machine, so the technical-support person can walk you through your problem.

Help Us Help You

Mark Williams Technical Support wants to help you fix your problem as quickly as possible. You can help us to help you by doing the following:

Before you contact Technical Support, *write down* as carefully as possible what you did that triggered the problem. Copy down exactly any error messages that appeared on the screen.

If the problem is triggered by a script or program, try to edit the script or program to the chunk of code that triggers the problem. The smaller the chunk of code, the better.

In your message, please include the following information:

- The make of your computer, and the type and clock speed of its microprocessor.
- The amount of RAM that you have.
- The size and make of your hard disk, and the make of its controller.
- The make of your display (i.e., tube) and controller card, and the amount of video RAM it has, if known.

If you have found an error in the manual, please mention the page on which the error occurs.

This information will help us to to clear up your problem as quickly as possible.

Where To Go From Here

If you have not yet done so, you should now install X Windows for COHERENT onto your system. For directions on how to do so, turn to the release notes that came with this manual and follows the directions given there.

The release notes also give errata — that is, mistakes in this manual that were discovered since its publication, and problems with X Windows for COHERENT that remain unresolved. We suggest that you note these in the appropriate places in this manual, and we hope you will accept our apologies for any inconvenience they may have have caused you.

After you have install X Windows for COHERENT, you should turn to the part of the manual that is most appropriate for your level of experience:

- If you are a beginner, turn to the following tutorial, entitled *Using the X Window System*. This walks you through X Windows, and teaches you how to use the window manager **twm**.
- If you are experienced with the X Window System, turn to the Lexicon articles entitled **X utilities** and **X applications**. These summarize the utilities and applications that are included with X.
- If you are interested in porting code to X, turn to the chapter entitled *Porting to X*. This gives practical advice on how to import X code, and walks you through the modification and compilation of a sample X application.

Using the X Window System

This chapter introduces you to the using the X Window System. This and the following chapter are intended for novices to X Windows. If you are experienced at using the X Window System, we suggest that you look at the Lexicon articles **X clients** and **X utiities** to see what programs are included with this package.

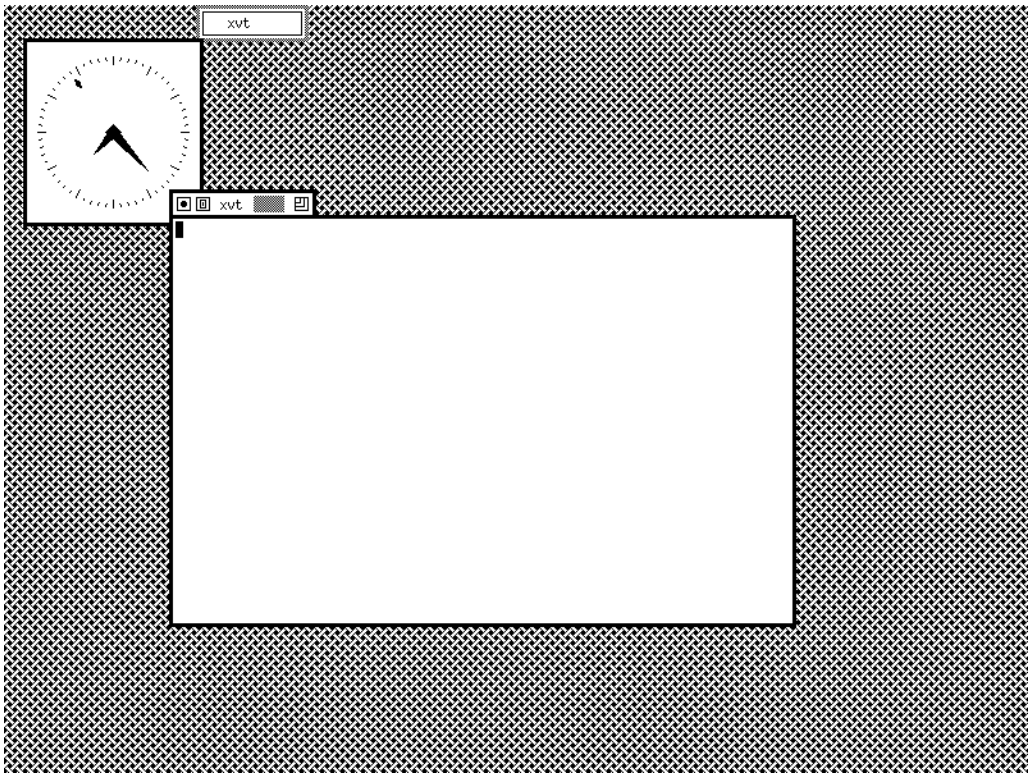
This chapter first introduces a graphical-user interface (GUI) and the window manager **twm**. The window manager is responsible for the appearance of the screen, and manages windows and icons. This portion of the chapter assumes that you have little or no experience with a GUI. Even if you have worked with other GUIs, such as Microsoft Windows, you should look over this section to see how **twm** differs from what you are used to.

This chapter then gives a tour of the X Window System, to show you what files are stored where. It then discusses the structure of the X Window System and the elements that comprise it. Some of this is a little dry for a novice; however, knowing something of what X can do and how it works will make it much easier for you to learn how to work with X.

Finally, this chapter ends with a description of how to modify the window manager to suit your needs and tastes.

Introduction to a Graphical Interface

To bring up X, log into your system's console, and type **startx**. If you have installed X properly, your screen will clear and within a minute your console will appear something like this:



The actual amount of screen space that the windows take up depends upon the mode of your physical screen and the size of the virtual screen, as described in the installation notes.

As you can see, there are three windows on the screen: a window that shows a clock, in the upper left corner of the screen; a small window at the top of the screen that displays the word **xvt**; and a large, blank window that has a prompt in it. The rest of the window is filled with a gray field.

Using the Mouse

To begin, try working with the mouse. The mouse should turned so that the wire on the mouse is away from you and the buttons are under your fingertips. As you slide the mouse "up" on the desk (that is, away from you), watch the screen. You'll see a small cursor moving "up" on the screen — that is, toward the top of the screen. Now, slide the mouse toward you. The cursor moves toward the bottom of the screen. Slide the mouse to the right — the cursor slides to the right; slide it to the left — the cursor moves to the left. The principle is obvious: the cursor moves around the screen in a manner analogous with the movement of the mouse. This cursor is called the *mouse cursor*; in X nomenclature is also called the *pointer*.

Notice that as the cursor moves around the screen, it changes shapes. When it is against the gray background, it is an × shape; when it is in the clock window or the little window labelled **xvt** it is an ←; while when it is the large, blank window its shape resembles a capital I. (For the sake of brevity, we will refer to the large, blank window as the **xvt** window. We'll discuss below just what **xvt** is.)

Also, as the cursor enters and leaves windows, things happen: when the cursor enters the **xvt** window, its border changes color, the text cursor changes from outlined to solid, and the inner border of the small window at the top of the screen also changes color. The same things happen when you shift the cursor into the small window, which suggests that the two windows are somehow linked.

As you've noticed, your mouse has buttons on it. (Some mice have two buttons, some have three. For present, we'll ignore the middle-mouse button and work only with the left and right buttons.) By pressing the mouse button, you can tell an application to do something; what happens depends on what programs you are running under X.

There are two kinds of stroke on a mouse buttons: *clicking* and *dragging*. Clicking is when you rapidly depress and release the button, just like you press a key on your keyboard. You often will use this stroke to press a "button" on the screen. Dragging is when you depress a mouse button, slide the mouse, and then release the mouse button. You often use this stroke to select an item from a menu or to move or resize a window. (How you can do these actions is described below.)

To try this, slide the mouse so that the mouse cursor shifts into the little window at the top of the screen. The cursor is in the little window when the window's outline brightens. Click the left-mouse button. The **xvt** window disappears from the screen, and a small, stylized 'X' appears in the little window. Without moving the cursor, click the left-mouse button again. The **xvt** window reappears, just as it was before, and the stylized 'X' disappears.

To explain what just happened, the small window is an *icon* for the **xvt** window. Icons give you a way to run a large number of programs simultaneously, without cluttering the screen with windows. You can specifically exclude certain X programs from having an icon. You would do this because an icon (like every other X object) consumes memory, and some programs (such as the clock program) always appear on the screen. We'll discuss icons further below.

Moving and Resizing Windows

With the mouse you can move and resize a window. Both involve dragging the mouse.

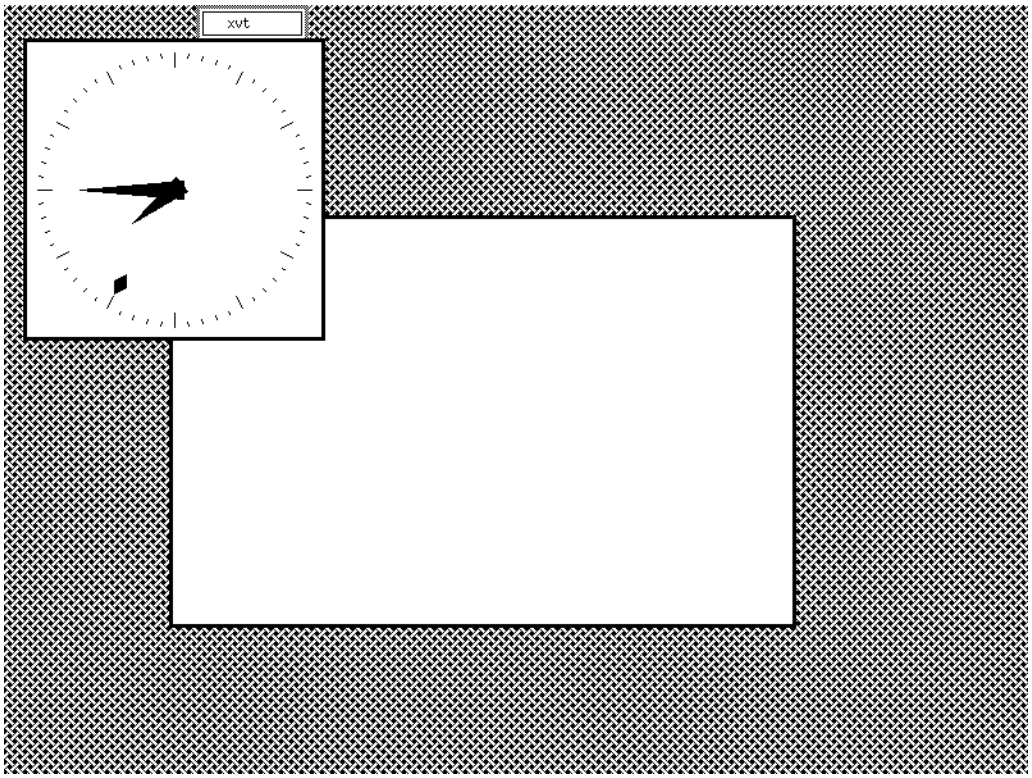
First, try to resize a window. Slide the mouse so that its cursor is touching the right or bottom border of the window. When the cursor touches the frame, it changes shape, from an × to an ←. Now, press the left-mouse button and hold it down. When you do this, three things happen:

1. The mouse cursor changes to a stylized '+' sign.
2. The clock window is covered with a grid.
3. A small window pops up in the upper left corner and displays a message that gives the X and Y dimensions of the window.

(If nothing happens, the tip of the ← is not touching the window's frame but is in the inner part of the window. Release the left-mouse button, slide the cursor until the tip of the ← is touching the frame, and then press the left button again.)

The numbers in the small pop-up window give the dimensions of the window, in pixels. (A *pixel* is one of the tiny dots that comprise your screen.) While holding down the left-mouse button, slide the mouse to the right and down until the numbers in the small window are approximately **230**×**230**. (We say "approximately" because a mouse is not a precision instrument, and trying to position it exactly can be a maddening task.) Note that as you drag the

mouse, the grid that was superimposed on the window "stretches" to match the movement of the mouse cursor. Now release the left-mouse button. The clock window is now larger, and your screen looks something like this:

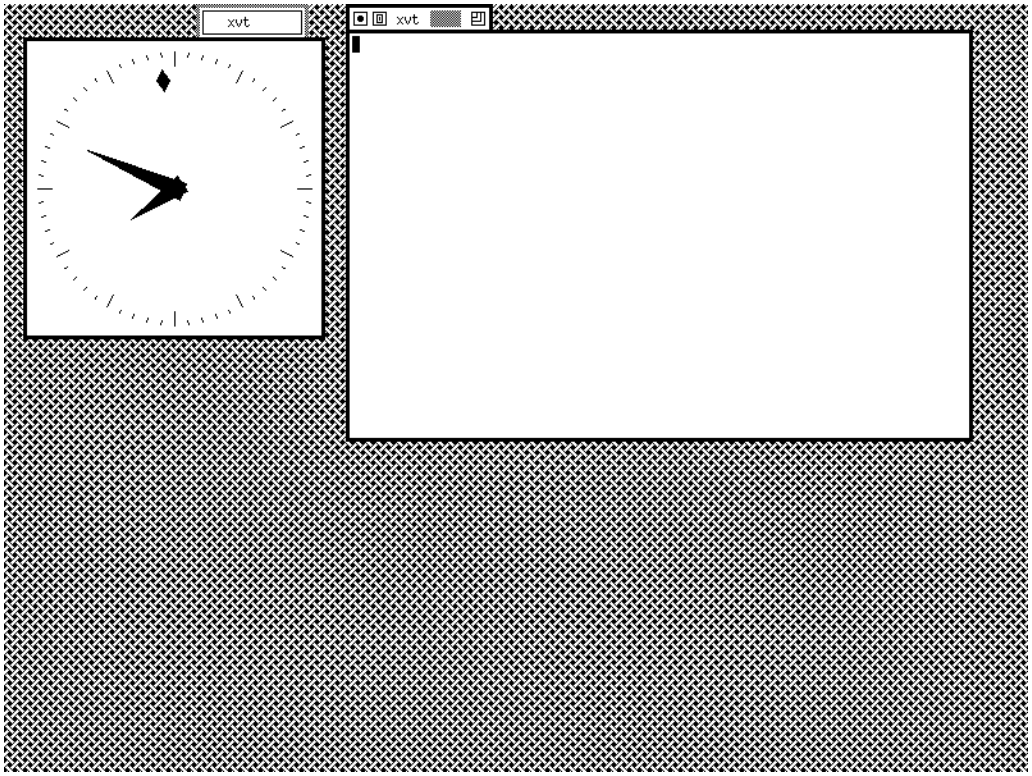


The clock window is now larger, the image of the clock has expanded to fill the larger window, and the clock window now overlaps the **xvt** window.

Now, try moving the **xvt** window. First, however, we need to shift the **xvt** window from behind the clock window (in X nomenclature, this is called the *background*) to in front of the clock window (in X nomenclature, the *foreground*). Move the mouse cursor into the icon and click the left-mouse button: the **xvt** window vanishes from the screen. Now, click the icon again: the **xvt** window appears again, but this time in the foreground, superimposed upon the clock window.

Now, slide mouse cursor into the **xvt** window. When the cursor enters the window, observe the area at the top of the window that has the word **xvt** in it — this area is called the *title bar*. When the cursor enters the **xvt** window, a small, gray patch appears in title bar. The other shapes in the title bar are *screen buttons*. A screen button is not a physical button, like the button on your mouse; however, it works like physical buttons in that you can "press" it by moving the mouse cursor to it and then clicking a mouse button.

Slide the mouse cursor so that it is touching in the title bar, but not touching any of the buttons. (The gray patch is part of the title bar, not a button.) Now, press and hold down the left-mouse button. Once again, the mouse cursor changes to a stylized '+'. While holding down the left-mouse button, drag the mouse up and to the right; while you slide the mouse, an outline of the window follows. Keep sliding the mouse until the **xvt** window is positioned to the right of the clock window; then release the left-mouse button. When you release the mouse button, X redraws the **xvt** window in its new position. Your screen should now appear something like this:



Screen Buttons

As we noted earlier, the little drawings in the title bar are screen buttons. Moving the mouse cursor to a screen button and clicking the left-mouse button tells X — or, to be more precise, the screen manager **twm** — to do something, just like pressing a button on your telephone.

The window manager displays three buttons on the title bar of each client's window. The following describes them in the order they appear in the title bar, from left to right:

1. The leftmost button, which has a bullet '•' in it, "iconifies" the window. That is, it erases the window from the screen and draws the stylized 'X' in that window's icon. The window will not appear on the screen until you "de-iconify" it by clicking on its icon.

Try this. Slide the mouse cursor so that it touches the leftmost button in the **xvt** window's title bar. When it touches the button, the cursor changes to a ☞, except pointing to the left. Click the left-mouse button: this "presses" the screen button. The **xvt** window vanishes, as before; it is now iconified. Now, slide the mouse cursor to the icon, and click the left-mouse button again. The **xvt** window is de-iconified, and reappears on the screen.

2. The second button from the left, which has a stylized 'D' in it, displays a drop-down menu. We will discuss this button below, when we introduce menus.

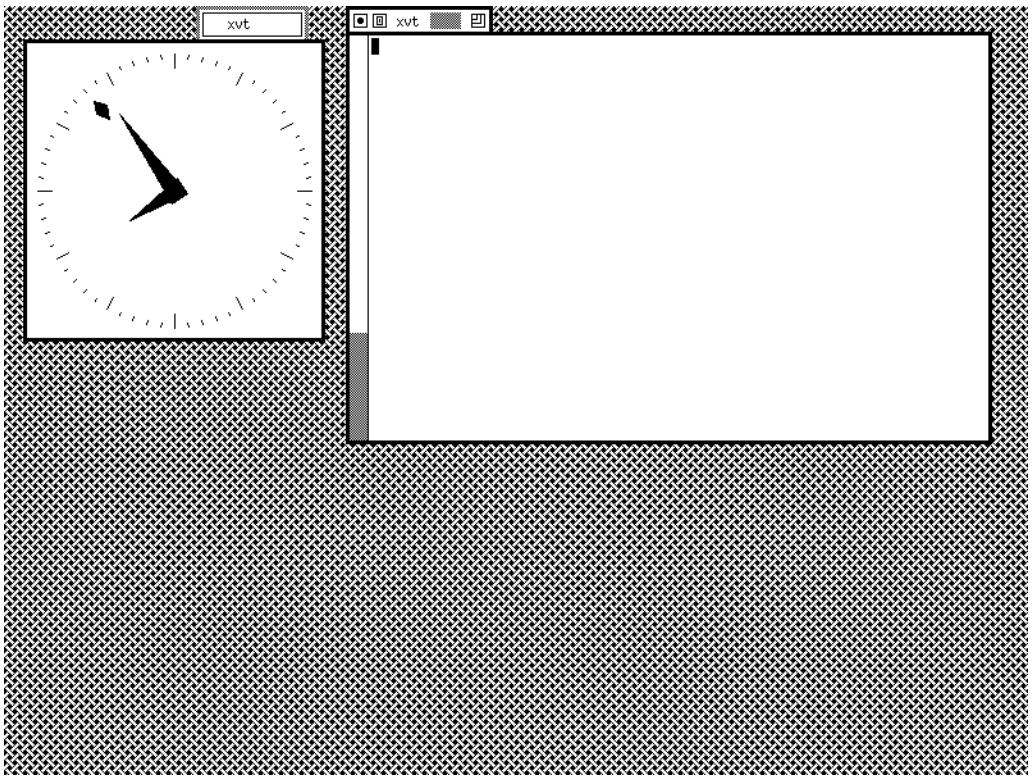
3. The rightmost button, which has a staggered set of squares in it, lets you resize the window, just as if you had clicked on the window's frame. To resize the window, move the mouse cursor to this button; press the left-mouse button and, while holding down the mouse button, slide the mouse cursor to the right and down, until the window is sized to suit your preferences.

Note that by default **twm** does not shrink a window until you have first moved the mouse cursor past the window's current borders. Thus, if you want to shrink a window, you have to first stretch the grid out past the current borders of the window, then slide it back in.

Many applications use screen buttons to gather information from you, but you operate each in the same way: move the mouse cursor to it and click the left-mouse button.

Sliders

If you look at the following figure, you'll see something different along the left edge of the **xvt** window:



The object to the left of the window is a column, with a gray patch at the bottom. This column is called a *slider*. You can use the mouse to move the gray patch up or down. By moving the slider, you change the value of a variable that is associated with the slider.

To change the value of a slider, move the mouse cursor onto the gray patch, then press the middle-mouse button and hold it down. (If you have a two-button mouse, click both buttons simultaneously and hold them down.) Then drag the mouse up or down on your desk; the gray patch follows the mouse cursor up or down. When you release the mouse buttons, the gray patch stays fixed where you dragged it; the corresponding value will have been raised or lowered by a proportionate amount.

Clients often use sliders to let you set a range of integral or ratio values. For example, **xvt** uses the a slider to let you set display lines of text that had scrolled off the top of the window earlier, up to a maximum of 64. By dragging the slider up or down, you can review text that had been displayed in the window, but no longer fits within the window — in effect, you are choosing which line will be the first line in the display.

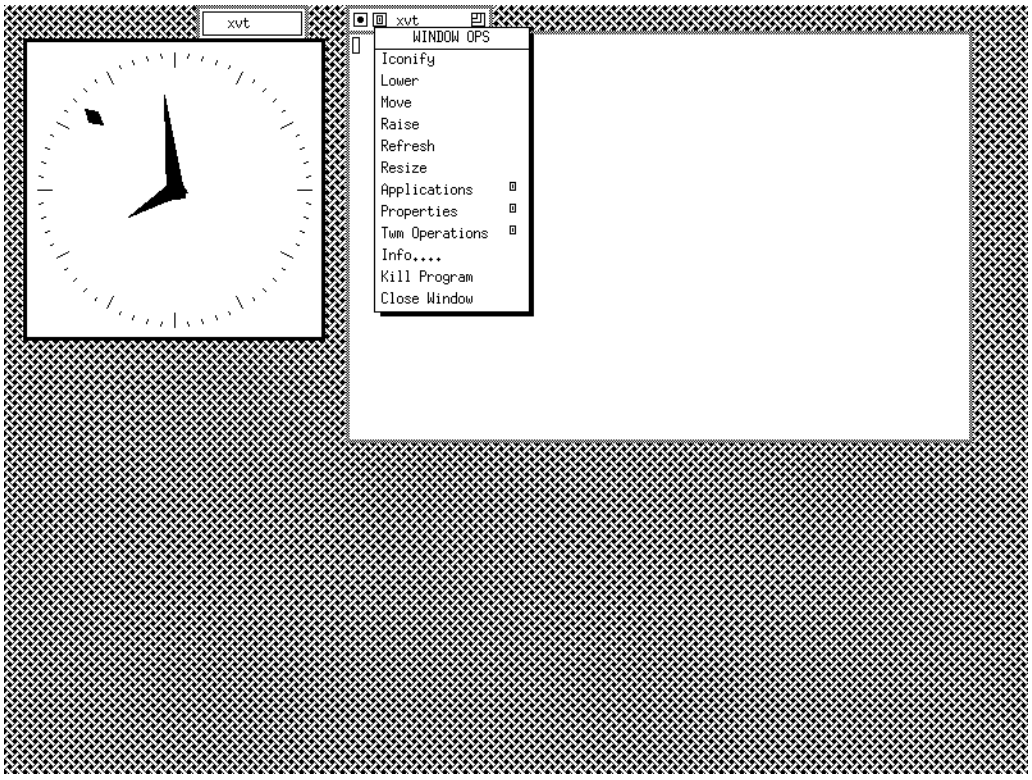
Many applications use a slider or sliders to set numeric values of this sort. Sliders will pop up again and again as you work with X.

Menus

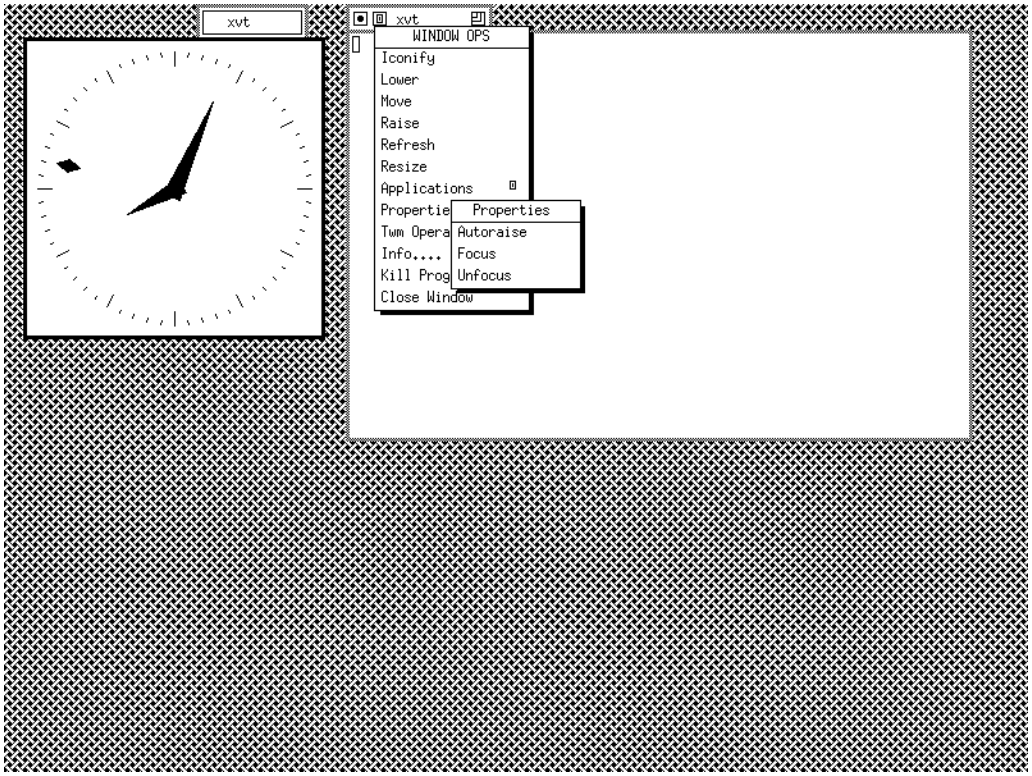
A *menu* is a list of entries, like a menu in a restaurant. You can select one of the items on the menu; the program then performs the action that corresponds to the item you have selected.

Many applications use menus. The window manager **twm** has three menus built into it. This section describes how to invoke these menus and their contents.

The first menu that we will describe is the one that appears when you press the screen button that is second from the left in a window's title bar. (This is the button with the stylized 'D' in it.) To invoke this menu, slide the mouse cursor to this button in the title bar for the **xvt** window; then press the left-mouse button and hold it down. The window now appears something like this:



As you can see, the menu has 13 entries. The entry at the top is the menu's title. If you look carefully, you can see that three of the entries have a small button on the right. The button indicates that that entry introduces a sub-menu; a sub-menu pops up if the mouse cursor simply touches the button — you don't have to press a mouse button (which is fortunate, as you are already holding down the left-mouse button). For example, if the mouse cursor touches the button in the menu entry labelled `(Properties)` a three-item sub-menu drops down, and the screen appears something like this:



The contents of this sub-menu will be discussed below.

The following describes each entry in the **WINDOW OPS** menu:

`(Iconify)`
Iconify this window. This is the same as clicking the leftmost button on the title bar.

`(Lower)` Lower this window into the background — that is, if this window overlaps with any other window, draw that window atop this window.

`(Move)` Move this window. This works exactly as if you had clicked on the title bar.

`(Raise)` Raise this window into the foreground — that is, if this window overlaps with any other window, draw this window atop that window.

`(Refresh)`
Redraw the window. Do this if any "junk" has appeared to "mess up" the window.

`(Resize)`
Change the size of the window. This works exactly the same as if you had clicked on the window's frame.

`(Applications)`
This entry has a button that, when touched by the mouse cursor, drops a sub-menu that shows a selection of applications that you can run under X. This menu will be discussed later.

`(Properties)`
This entry has a button that, when touched by the mouse cursor, drops a sub-menu that lets you change some of **twm**'s properties. The term *property* has a precise meaning under X, and will be discussed later; for now, consider a property as being a feature of an object that affects the object's appearance or behavior. The **Properties** sub-menu has the following three entries:

`(Autoraise)`
If you turn on this property, the window manager raises this window to the foreground whenever the mouse cursor enters it. To turn off this property, simply select this sub-menu entry again.

(Focus) Focus the window manager's attention on this window. This means that the window's frame is always bright, even when the mouse cursor is not in this window. Further, whatever you type on the keyboard appears in this window, regardless of where the mouse cursor is on the screen.

(Unfocus)
Stop focusing the window manager's attention on this window. This is the default.

(Twm_Operations)
This menu entry drops a long sub-menu that lets you invoke many features of the window manager **twm**. This menu and its contents are discussed below.

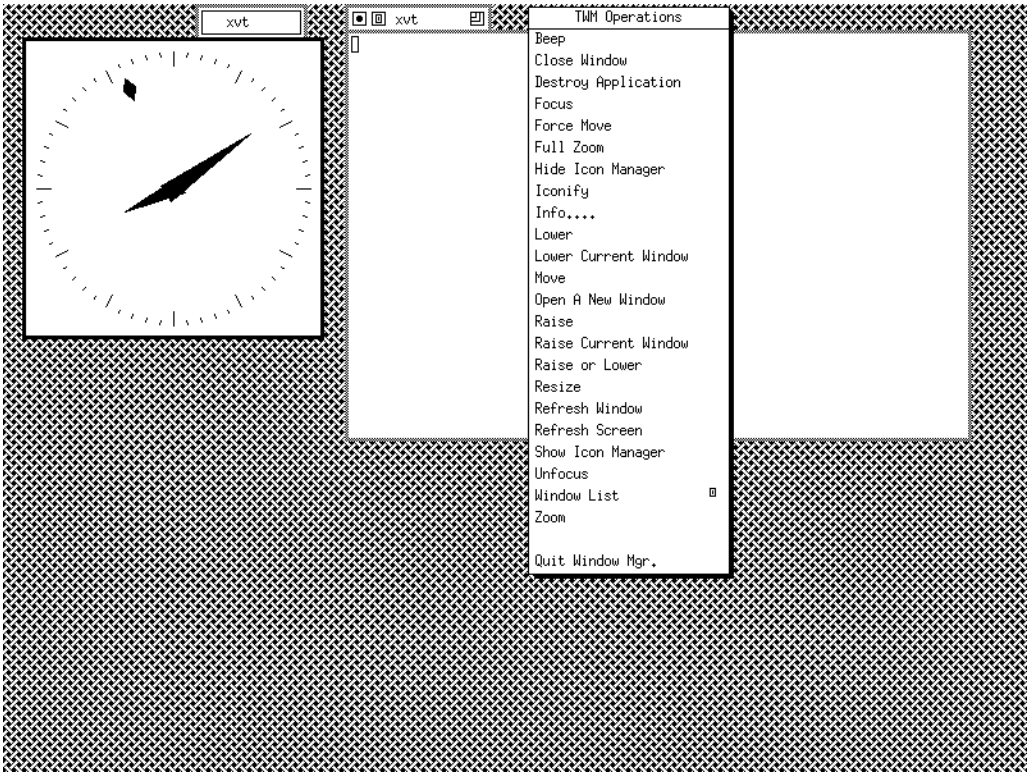
(Info...)
This menu entry opens displays information about the current window: its name, position on the screen, and other information.

(Kill_Program)
This kills the current window dead, just the same as if you had typed the command **kill -SIGKILL** for this program. Do *not* use this option unless you absolutely must kill an application, because a program that dies in this way often leaves memory and the file system littered with debris.

(Close_Window)
This is a kinder, gentler way of killing the program in this window. It closes this window; in theory, an application perishes gracefully when its last (or only) window is closed. Unfortunately, this does not work for every application, and in some cases you may be compelled to resort to the heavy, blunt object of the previous menu entry.

To see how a menu works, try selecting an item from it. Click the title-bar button that drops this menu; then, while holding down the left-mouse button, drag your mouse down until the entry (Iconify) is highlighted. Then release the mouse button. This invokes the "iconify" feature, and "iconifies" the window. Now, click the **xvt** icon at the top of the screen to redisplay the **xvt** window.

As we mentioned above, the menu entry **Twm_Operations** drops a menu of things that you can do to, and with, the window manager **twm**. The same menu appears if you move the mouse cursor so that it is positioned over the screen's background (when changes shape to an ×) and press the right-mouse button. Try it; your screen should appear something like this:



If the physical screen is smaller the virtual screen, you won't be able to see all of the menu at once. To see the bottom of the menu, keeping sliding the mouse cursor down; the borders of the physical screen will shift downward to keep the mouse cursor in view.

The following describes the entries in the **TWM Operations** menu:

`(Beep)` Ring the bell.

`(Close_Window)`
Close the current window. This is the same as the `(Close_Window)` entry in the previous menu.

`(Destroy_Application)`
Kill an application. This is the same as the `(Kill_Program)` entry in the previous window.

`(Focus)` Focus the window manager's attention on a window. If you invoked the **TWM Operations** menu from within the **WINDOW OPS** menu, **twm** fixes its focus on the window from within which you invoked **WINDOW OPS**. If, however, you invoked **TWM Operations** by clicking the right-mouse button, then **twm** lets you pick the window interactively: it changes the mouse cursor to a bull's-eye; move the mouse cursor to the window upon which you want **twm** to focus, and click the left-mouse button. **twm** will focus on this window until you go through this procedure again for that window, or invoke the **Unfocus** menu entry, below.

`(Force_Move)`
Move a window, even if the window's properties specifically exclude its being moved.

`(Full_Zoom)`
Expand a window to fill the entire screen. The window that **twm** "zooms" is determined just as with the `(Focus)` entry, above.

`(Hide_Icon_Manager)`
Erase, or "hide," all icons from the screen. The icons are kept in memory, waiting for you to "un-hide" them.

(Iconify)
Iconify a window. The window that **twm** iconifies is determined just as with the (Focus) entry, above.

(Info...)
Display information about a window, such as its name, size, and position on the screen. The window about which **twm** displays information is determined just as with the **Focus** entry, above.

(Lower) Lower a window. Again, the window that **twm** lowers depends upon how you invoked this menu.

(Lower_Current_Window)
This lowers the window that was raised last.

(Move) Move a window. This works just as if you had clicked on a window's frame.

(Open_A_New_Window)
This invokes the application **xvt** to open a new window and run your shell of choice in it. Yes, this is the same **xvt** whose window you have manipulated so far. This application is introduced below; for detailed information on it, see its entry in the Lexicon at the back of this manual.

(Raise) Raise a window. Again, the window that **twm** raises depends on how you invoked this menu.

(Raise_Current_Window)
Raise the window that was lowered last.

(Raise_or_Lower)
Select a window interactively: if it is lowered, raise it; if it is lowered, raise it.

(Resize)
Resize a window. This works just as if you had clicked on a window's title bar. Again, the window that **twm** resizes depends on how you invoked this menu.

(Refresh_Window)
Refresh a window.

(Refresh_Screen)
Refresh every window on the screen. **twm** blanks the screen briefly, then redraws it.

(Show_Icon_Manager)
Redisplay the icon manager, if it is hidden.

(Unfocus)
If **twm** is focused on a window, unfocus it. If **twm** is not focused on a window, do nothing.

(Window_List)
This entry has a button on it. If the mouse cursor touches this button, a sub-menu appears that lists every window **twm** is handling at this moment, whether or not it is iconified or hidden.

(Zoom) Expand a window to fill the window vertically; do not, however, change its horizontal dimension.

(Quit_Window_Mgr.)
Finally, shut down the **twm** window manager. This also shuts down the X server, and returns control to the ordinary COHERENT console interface.

So far, we have discussed **twm** and how to manipulate windows: how to move them, change their size, change their focus, hide them, redisplay them, and so on. Now we will discuss just what's going on within the windows themselves — which is, after all, the whole point of the X Window System.

Introduction to xvt

The window we've called the "**xvt** window" is the window for an X program named **xvt**. This program mimics a VT-100 terminal. COHERENT "thinks" that this window is another terminal, just as if you had plugged a dumb terminal into a serial port on your machine.

When **xvt** opens its window, by default it invokes a shell through which you can give commands to COHERENT. You can invoke other X clients via **xvt**, dial out to another system, edit files, and play games.

You can open more than one **xvt** window on your screen if you wish. The number of windows you can manage comfortably depends upon the size of your screen; if you have a small screen, you may wish to iconify the **xvt** windows that you are not using at the moment. If your system were on a network, you could open a number of different windows, each logged into a different machine on the network. In this way, users who are hooked into the

Internet can simultaneously view and manipulate data from machines that are thousands of miles apart.

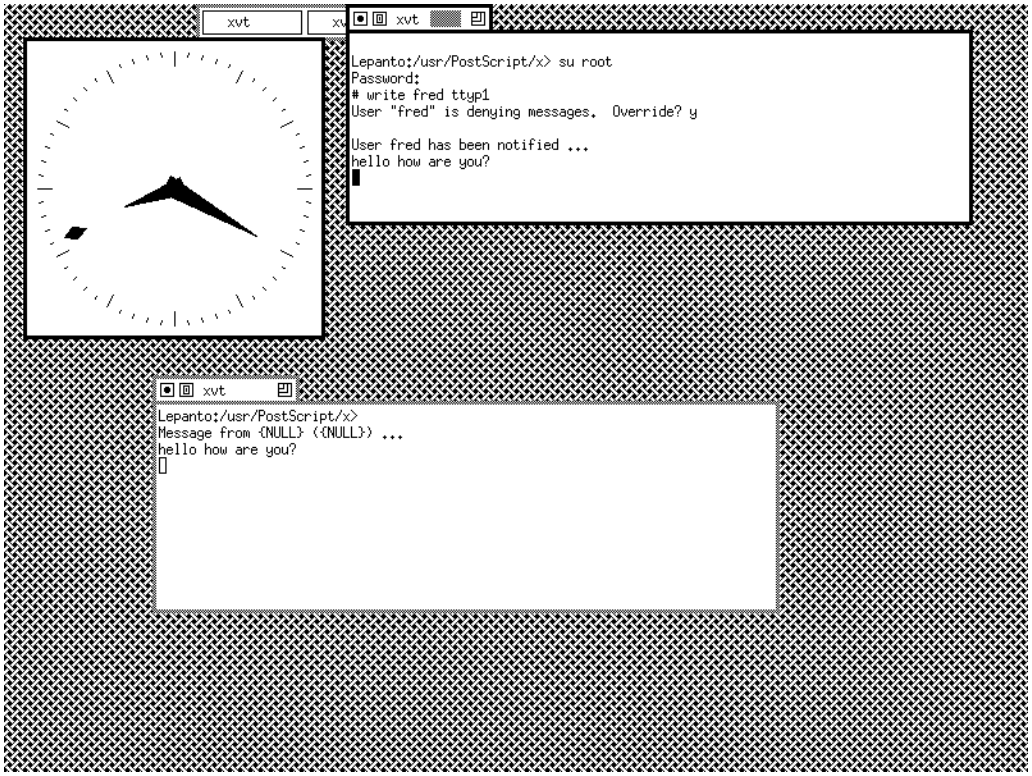
To give COHERENT a command via **xvt**, simply move the mouse cursor into the **xvt** window (so that X knows that what you type at the keyboard is intended for this window), and type the command as you always do. To spawn another terminal window, simply type:

```
xvt &
```

In a moment, X draws the outline of an **xvt** window, and fixes it to the mouse cursor. Slide the mouse until the window outline is where you want the new window to be, then click the left-mouse button. The new terminal window appears, with a COHERENT prompt in it. You can shrink the new terminal window so it and the original terminal window fit together onto your screen; or you can iconify it, or use the **Raise** and **Lower** entries from the **twm** menus to help you work with both windows.

Note that when **xvt** opens a window, the window is set to 80 characters wide by 25 rows high, just like an ordinary terminal's screen. Most COHERENT applications expect that size of a screen, and have no idea that they are talking to a window that can be resized. Thus, if you shrunk an **xvt** window, you can expect that some visually oriented COHERENT applications — such as MicroEMACS, **vsh**, or **chase** — will not work correctly. Note, too, that if you kill an **xvt** window, any program that you invoked from it — either a COHERENT application or an X client — dies as well. (The COHERENT command **nohup** will help you get around this limitation; for details, see its entry in the COHERENT Lexicon.)

The following screen shows one unusual application of multiple **xvt** windows: the user here is conversing with himself in two different windows via the COHERENT utility **write**:



Cutting and Pasting

X has a built-in facility with which you can cut and paste text. This facility is impractical for moving large amounts of text, but is handy for copying a command or a set of information from one window into another, or into a file. Note that this facility only works with text — you cannot use it to copy a bit-mapped image or an icon.

To cut text, move the mouse cursor to the point where you want to start cutting, then press the left-mouse button. Drag the mouse to the end of the block of text you wish to cut; then release the left-mouse button. The X server copies the cut text into both its cut buffer and into its primary selection (that is, the property **PRIMARY**).

Another way to cut text is to click the left-mouse button at the beginning of the block of text you wish to cut, then move the mouse cursor to the end of the block and click the right-mouse button. All text between the two clicks is highlighted, to show that you have cut it.

To un-cut text, move the mouse and click the left-mouse button again. The text will be un-highlighted, to show that it is no longer cut.

To paste text, move the mouse cursor to the spot where you want to "drop" the text, then press the middle-mouse button. (If your mouse has only two buttons, press both buttons simultaneously.) X handles the pasted text just as if you had typed it from the keyboard: if the text is pasted into a window that cannot accept keyboard input, then the pasted text goes into the bit bucket. To drop the cut text into a file, invoke an editor or the COHERENT command **cat** in an **xvt** window, then drop the text into that window. The program you invoked in the window will save the text into a file, just as if you had typed it by hand.

Introduction to xclock

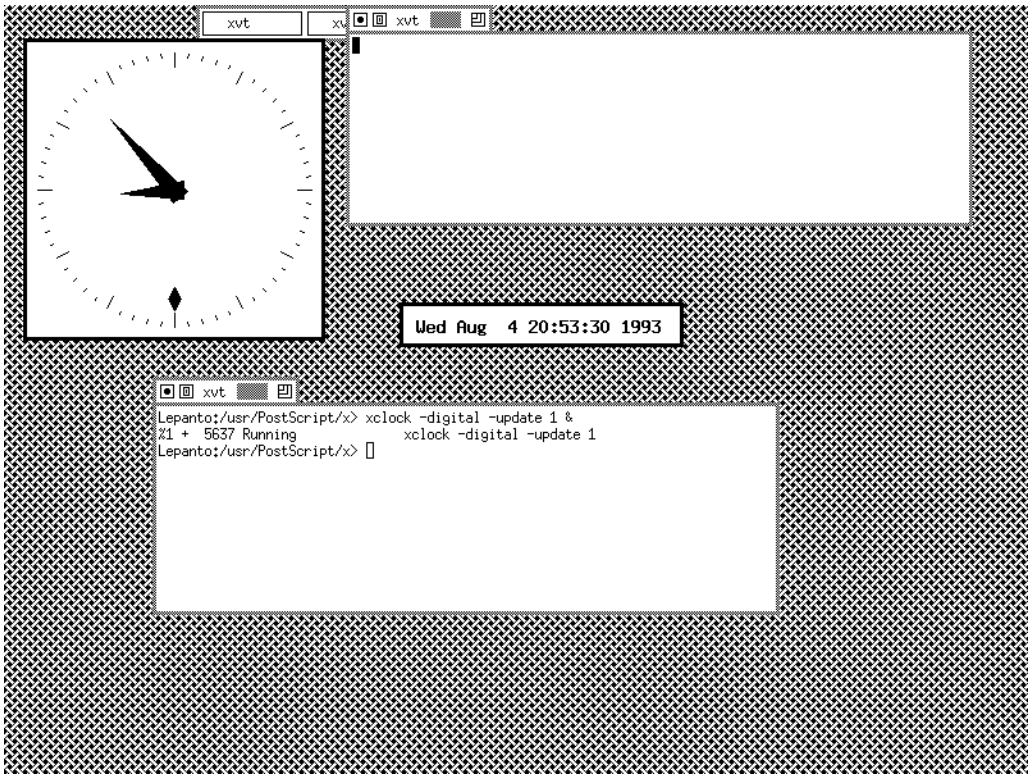
The window that we've called the "clock window" is the window for an X program named **xclock**. This program has many options, some of which you can invoke via command-line options, like any other COHERENT program.

For example, **xclock** by default displays an analogue clock — that is, a clock with a dial and hands. You can, however, request a digital clock: to do so, just type the following command into your **xvt** window:

```
xclock -digital -update 1 &
```

The option **-update 1** tells **xclock** to update itself every second. The default is every 60 seconds.

Once again, a small outline of a window appears. Slide the mouse until the window is placed where you want, then click the left-mouse button. You now have a small digital clock, which displays the date as well as the time:



To remove your new clock, select the command `(Close_Window)` from the menu **TWM Operations**.

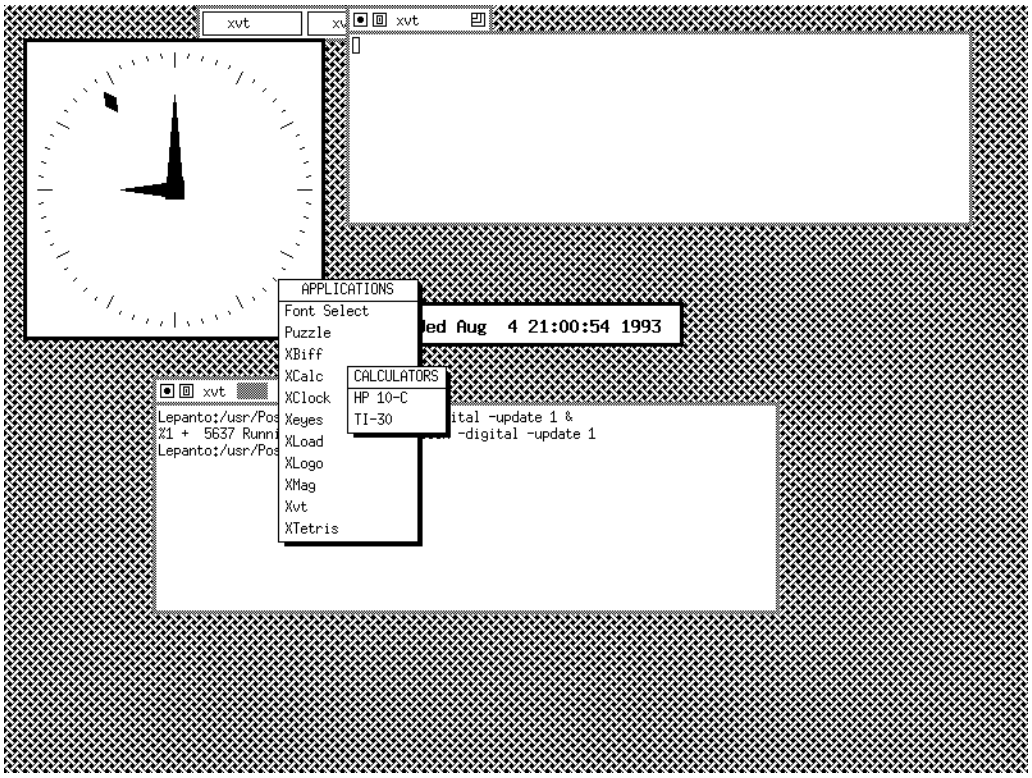
TUTORIALS

For a full list of options to **xclock**, see its entry in this manual's Lexicon.

The Applications Menu

Earlier, we noted that **twm** has three menus available to you. We have described two: **WINDOW OPS** and **TWM Operations**.

The third menu is **APPLICATIONS**. Through it, you can invoke some of the more commonly used X clients, without having to type a command into an **xvt** window. (This explains, among other things, how you can invoke an **xvt** window when the screen has no **xvt** window into which you can type the command.) This menu appears when you position the mouse cursor over the background of the screen (as shown by the fact that the mouse cursor changes to an ×), and click the left-mouse button. The following figure shows the menu **APPLICATIONS**:



APPLICATIONS contains the following entries:

(Font_Select)
: Invoke the X client **xfontsel**, which helps you to select a font interactively. The description of fonts under X is something of a "black art"; for details, see below or see the entry for **xfontsel** in the Lexicon.

(Puzzle)
: Invoke the X client **puzzle**, which lets you play a scrambled-tiles game.

(XBiff)
: Invoke the X client **xbiff**, which displays an icon of an old-fashioned roadside mailbox. When you receive new mail, the flag on the "mailbox" pops up.

(XCalc)
: Invoke the X client **xcalc**, which displays the image of scientific calculator — either a Hewlett-Packard or a Texas Instruments model, whichever you prefer. By clicking on the image's buttons, you can invoke all of the features of a real scientific calculator. As you can see from the previous figure, this entry invokes a sub-menu, from which you can select the type of calculator you want.

(XClock)
: Invoke the client **xclock**, described above.

(Xeyes) Invoke the X client **xeyes**. This client draws on the screen a pair of eyes whose pupils follow the mouse cursor around the screen. This can help you find the mouse cursor on a cluttered screen.

(XLoad) Invoke the client **xload**. This client displays a histogram that graphs the load on your system. You can use this client to gain an idea of how many cycles a given program consumes.

(XLogo) Invoke the X client **xlogo**, which merely displays the X logo. This logo is the stylized X that appears in an icon when you have iconified a window.

(XMag) Invoke the X client **xmag**. This client magnifies part of the screen, to help you see how a shape is built out of individual pixels.

(Xvt) Invoke another **xvt** window.

(XTetris) Invoke the X client **xtetris**, which plays the popular game Tetris.

Each of these applications is invoked with selected command-line options; for example, **xclock** is invoked to display an analogue clock rather than a digital one. Below, we will show you how to change these defaults.

Shutting Down X

Shutting down X is a two-stage operation: first, you must close all **xvt** windows, and then you must kill the server.

To shut an **xvt** window, simply move the mouse pointer into the window (so that its frame is highlighted), then type the command **exit**. If nothing appears to be happening, wiggle the mouse a bit; X often waits for you to do something before it redraws the screen to reflect the effects of any commands you may have issued.

Once all **xvt** windows are closed, you can shut down the X server. There are two ways to do this.

The first way is to do so via a menu. Invoke the menu **TWM Operations** by moving the mouse cursor over the screen's background, then press the right-mouse button. Drag the mouse until the last entry in the menu, which is labelled (Quit_Window_Mgr is) highlighted. Then release the right-mouse button. All windows close, the X server turns itself off, and you are returned to the standard COHERENT character-based interface.

The other way is simply to type **<ctrl><alt><backspace>**. No matter when you type this combination of keys, the X server will detect it and shut itself down.

The two methods vary in how they flush buffers and tidy up after themselves. In general, it is better to shut down in a controlled fashion, through the menu entry, than by pressing the magic combination of keys.

Note that you can kill the X server without closing your **xvt** windows. However, this is not a good idea because COHERENT may "think" that you are still logged into an **xvt** window, even though that window no longer exists. This may create problems for you at some future time.

Conclusion

This concludes the introduction to the **twm** screen interface. We have introduced the basic "vocabulary" of the X GUI, discussed **twm** and its menus, and shown how you can invoke and operate some other X clients under **twm**.

The following section gives you a "nickel tour" of the files and directories that comprise the X Window System. Subsequent sections introduce the internal structure of X. Knowing how X is structured internally will help you grasp how to configure X. The final section in this chapter describes how to configure **twm** to suit your preferences.

A Tour of the X System

The X Window System is installed into directory **/usr/X11**. The following describes the contents of each directory in the X Window System:

/usr/X11/bin
This directory holds the executable programs themselves. Most are created when you install your system from the object modules in **/usr/X11/objs**. For an introduction to these program, see the following chapter, *X Windows Clients*.

/usr/X11/doc
This directory holds public-domain documentation on the X Window System that you may find useful.

/usr/X11/include/X11
This directory holds header files and bit-mapped images that are included in the source code of an X application. It holds the following sub-directories:

Xaw Header files used with the Athena widget set.

Xmu Header files used with miscellaneous X utilities.

bitmaps
This holds the C source code for bit-mapped images that X Windows uses. You can include these images in programs that you import or write.

extensions
Header files for extensions to the X Window System.

sys System-level header files that are used with X Windows.

/usr/X11/lib
This directory holds libraries of X Windows routines. Please note that these libraries do *not* include code to use or emulate a mathematics co-processor.

This directory also holds configuration files for the X server, in particular **Xconfig**.

/usr/X11/lib contains the following sub-directories:

app-defaults
Files of settings, or "resources," for selected X programs. By modifying an program's resource file, you can modify its appearance and behavior without having to modify its source code. The next chapter describes how to do this.

config This holds files that describe how this implementation of the X Window System is configured. They are read by the utility **imake** to build a correctly configured **Makefile** from an **Imakefile**. For details, see the Lexicon entries for **imake** and **xmkmf**.

fonts This directory holds the fonts used by the X Window System. X Windows for COHERENT comes with two directories of fonts: **fonts/misc**, which holds miscellaneous fonts (such as the font used by the mouse cursor); and **fonts/75dpi**; which holds 75 dot-per-inch, proportionally spaced fonts. For information on how to display a font, see the Lexicon entry for **xfd**.

nls This holds tools used by X developers.

twm This directory holds the file **system.twmrc**, which is the default resource file for the **twm** window manager. **twm** reads this file if your home directory does not contain a file named **.twmrc**. If you want to customize **twm** to your liking, copy **system.twmrc** into your home directory, rename it **.twmrc**, and modify it. The following section describes how to modify **$HOME/.twmrc**.

x11perfcomp
This holds scripts used to help gauge system performance

xinit This directory holds the file **xinitrc**. The utility **xinit** reads this file if your home directory does not contain a file named **.xinitrc**. If you want to customize **xinit** to your liking, copy **xinitrc** into your home directory, rename it **.xinitrc**, and modify it to your liking. The following section describes how to modify **$HOME/.xinitrc**.

/usr/X11/objs
The object modules for the clients and utilities included with this package. When you install the package onto your system, they are linked with the libraries in **/usr/X11/lib** to create executable programs.

This concludes our nickel tour of the X Window System. The next section discusses the structure of X Windows, in preparation to discussing how you can modify X.

The Structure of X

This section introduces the structure of the X Window System. If you are a beginner, you may find some of this section to be a little dry. However, if you take a few minutes to read these descriptions, you probably will find it easier to learn how to customize the X Window System.

Background

To begin, a little history. What we call the *X Window System* was created at the Massachusetts Institute of Technology. The first edition, or *release*, of X appeared in 1984; release 11, which is the current release of X, appeared in 1987. Since 1987, release 11 has been revised several times, to fix bugs and widen its scope; the current edition of X is release 11, revision 5. (X documentation often uses a kind of shorthand to indicate the

release and revision for which a given program is intended, e.g., "X11R5" or "X11.5".)

MIT performed much of its work on X in partnership with Sun Microsystems; other computer manufacturers also contributed to X. The notes at the end of each Lexicon entry usually name the company or person who contributed the given program to X. The bulk of the X Window System is copyrighted by MIT; source code, however, is available for a minimal cost from MIT, and MIT does not demand a royalty for X. (Some packages — such as X Windows for COHERENT, which you are now using — contain code that is proprietary; therefore, do not assume that you can copy and give away your X software.)

Unlike other graphical user interfaces (GUIs), X makes no assumptions either about the operating system that is running underneath it, or about the manner by which the user communicates with the computer. This approach has a few disadvantages. In particular, it has led manufacturers to create a number of different user interfaces, of which the best known are Motif (created by the Open Software Foundation) and OPEN LOOK (created by Sun and AT&T). Thus, the appearance and behavior of an X system can vary greatly from one computer to another.

However, the disadvantages of this approach are greatly outweighted by its advantages. Because X makes no assumptions about its underlying operating system, X has been adopted as a *de facto* standard by a number of leading computer manufacturers. Although it usually runs with some version of the UNIX operating system (or a UNIX-like operating system, such as COHERENT), it has also been ported to limited operating systems like MS-DOS. Its design also makes X an ideal interface for a network of computers: even if each computer has a different architecture and operating system, all can appear the same to the user and run in more or less the same manner, thanks to X.

The flexibility of X also means that it is easy to port X programs from one operating system to another. Thus, as soon as X is brought up on an operating system, a wealth of software quickly becomes available to users.

X Architecture

The flexibility of the X system results from the fact that X carefully isolates into a single program each class of tasks that it must perform. For example, it isolates into one program, called the *server*, all of the work of driving the hardware. If an X application needs to draw something onto the screen, it simply invokes a feature of the server, which does the job. Thus, to port X to new operating system, one must rewrite the server; but once that is done, importing an X application often involves no more than recompiling it. (There are exceptions to this, of course, but on the whole an X application that "plays by the rules" can be ported with very little trouble.)

Therefore, X can be compared to a well-designed machine. Each part of the machine does one thing, and only one thing. If one part needs to perform a task that falls into the domain of second part, the first part sends a message to the second part and lets the second part execute that task for it. Any one part can be pulled out of the machine and replaced without affecting the performance of any other part. This way of fashioning a computer program is called *object-oriented program design* (OOP). Although the principles of OOP lie beyond the scope of this manual, you will find it helpful when you work with X to try to think in terms of objects and how they fit together.

The manner in which the parts of the X machine fit together is called its *architecture*. The architecture of X has three major parts: the *server*, the *client*, and the *window manager*.

server The server is the program that actually runs the X system. It does all the low-level work of drawing images on the screen; it manages communications among X programs; and on networked X systems, it manages permissions. Every other X program registers with the server when they come up; all of its work with the screen is done via requests to the server.

X Windows for COHERENT comes with two servers: one that supports color VGA for a set of the most popular VGA chip sets; and one that supports monochrome images on practically every variety of VGA or SVGA card. Most programs run without alteration on either server.

As a rule of thumb, there is one server for each computer at which a person works. Here, the term "computer" includes such devices as X terminals, as well as PCs and workstations.

clients A client is a program that registers with the X server, and uses the server's facilities to work with the user. As noted above, a client does not contain any code for manipulating hardware — it does not know how to talk to the mouse or the screen. All it does is send requests to the server and receive information from the server.

In this manual, we speak of two types of application: *clients* and *utilities*. In this manual, a client is something that let you do something under X, such as play a game or edit a file. A utility helps you to run X itself. Both clients and utilities, however, are X clients in the broad sense: both register with the server and do their work through it.

On a networked X system, a client can register itself with any server on the network, provided that the user who launched the client has permission to register with that server. For example, a person sitting at one workstation can invoke a client and tell it to register itself with the server on another person's workstation, to show that other person a document or graph. COHERENT does not yet support networking; when it does, you will be able to tie your machine into a network of computers and work together via X.

window manager

The *window manager* is a special client that manages the screen for all other clients. It allocates screen space to all other clients, and makes sure that no client intrudes on another client's space. It provides the tools with which you you can move a window or change its size. It also manages all parts of the screen that belong to no individual client; these include the background of the screen (also called the screen's *root window*), icons, window frames, title bars, and drop-down menus. The window manager is largely responsible for the "look and feel" of a given implementation of X.

In your day-to-day work you probably will not need to concern yourself with the architecture of X. Its parts fit together to make one smoothly running machine. However, you will find it helpful to bear the above descriptions in mind, because they will help you track down the cause of any problems that you may encounter.

Libraries, Widgets, and Resources

Internally, X has a hierarchical design. The lowest level of the hierarchy are the routines in the library **Xlib**. These routines do all primitive tasks, such as drawing lines and shapes on the screen, reading input from the keyboard, and polling the mouse. **Xlib** underlies all versions of X.

On top of **Xlib** sit *toolkits*. A toolkit is a set of routines that combine the low-level routines of **Xlib** into tools that applications can use directly. For example, a toolkit routine may combine a set of **Xlib** routines that draws shapes into one routine that draws a button. A toolkit sometimes includes its own low-level routines to cover tasks that are not addressed in **Xlib**; for obvious reasons, this is discouraged.

Numerous toolkits have been written for X. The tools in this package are built from the MIT toolkit **Xt**.

Atop a toolkit sits a *widget set*. A widget combines a set of routines with one or more bit-mapped shapes that can be displayed on the screen. For example, a button widget can display a bit-mapped shape, superimpose a button from the toolbox upon the shape, and include routines for polling the mouse and reacting to the mouse events (e.g., when a mouse button is pressed or released).

Widgets can be assembled to form "composite widgets." Thus, a widget set has a hierarchy within itself: whatever affects a widget that is lower in the hierarchy affects all widgets that are built atop it.

A screen interface, such as Motif or OPEN LOOK, consists of a toolkit, a widget set, and a screen manager that is built from the toolkit and widget set. The interface embodies a design of how the screen should appear and how the interface should behave.

X's widgets are designed so that you can change "on the fly" many of their aspects. For example, a button widget can let you set the text that appears in it, its color, its position on the screen, dimensions, whether it has rounded or square corners, and so on. You can do this by setting a widget's *resources*.

You can set resources in any, or all, of three ways:

1. First, many applications read information from command-line options. Applications that are built from the **Xt** toolkit take a standard set of command-line options, which (among other things) let you set colors, the window's geometry (that is, its initial size and position on the screen), and whether it appears in reverse video in reaction to an event. The suite of command-line options varies from application to application: Some have a rich set that lets you fine-tune the application, whereas others' are sparse.

2. Some applications read a file from directory **/usr/X11/lib/app-defaults**. In this file, you set resources for that application. For some applications, such as **xcalc** or **xfontsel**, the application file can be quite large and complex. The settings in an application's defaults file apply only that application. By modifying its defaults file, you can change an application's appearance and, in some instances, its behavior.

3. You can use the command **xrdb** to build a resource data base within the X server. Most applications read this data base as they come up. The contents of this data base override any defaults built into any application, or any resource settings they may have read from an application-defaults file. The settings in the resource data can apply either to an individual application or to all applications, depending upon how you "phrase" them.

As we noted above, widgets most often are built out of other widgets. If a widget is used to build other widgets, that first widget is said to be a widget *class*. X has a mechanism by which you can set a widget class; if you alter a widget class, every subordinate widget that incorporates the first widget into itself is altered as well.

The next chapter, which introduces the X clients, describes how to set widget classes. X widgets and widget classes are a large and complex topic of study; however, it is possible even for novices to make minor modifications to a defaults file, as we will show below.

Bit Maps

A *bit map* is a picture that is formed by turning cells in a grid on and off. X interprets each cell in the grid as a pixel. The mouse cursor, for example, is a bit-mapped picture.

You can draw a bit-mapped image with a ordinary text editor, using a pattern of pound signs '#' and hyphens '-', or you can use one of the tools that comes with X. You can then "compile" the bit map into C code that can be included in an application, just like any other header file.

X includes tools with which you can draw, edit, and compile bit maps. These are introduced in the next chapter.

X Windows for COHERENT comes with a selection of bit maps. These are stored in directory **/usr/X11/include/X11/bitmaps**.

Fonts

A *font* is a collection of shapes that comprise the letters of the alphabet and most commonly used punctuation marks. A font has 14 attributes that describe its size, weight, style, manufacturer, the character set it encodes, and other information.

The characters in a font are bit-mapped images. Thus, a "font" does not necessarily have to be the letters of the alphabet; for example, the mouse cursors are stored in a special font called **cursor**. (For a table of all available mouse cursors, see the entry in this manual's Lexicon for the X utility **xfd**.) This also means that, if you are so inclined, you can create and use your own fonts.

Fonts are stored in the directories named in the server's *font path*, which is set in the file **/usr/X11/lib/Xconfig**.

Because font files can be quite large, it is customary for them to be compressed so they take up less space on disk. X can use fonts in their compressed form; you do not have to uncompress a font for X to use it.

Each directory named in the font path must contain a file named **fonts.dir**. Each entry in this file gives a font's full, 14-part name and the file in which it resides. If you move a new font into a font directory, you must modify this file or X will not be able to find the font.

The X utility **mkfontdir** reads all of the fonts in a directory and rebuilds that directory's **fonts.dir** file to show what fonts that directory actually contains.

A font's full name describes all 14 of a font's attributes. For example, the following gives the full name of the font used for on titles:

```
-adobe-times-bold-*-normal-*-*-140-*-*-*-*-*-*
```

For the sake of brevity, you can use an alias for a font's full name. A font directory should also contain a file named **fonts.alias**; this file maps aliases to full font names.

X Windows for COHERENT comes with a selection of fonts for your applications. It also comes with tools for selecting fonts interactively, and for modifying a **fonts.dir** file. These are introduced in the next chapter. For more information on selecting fonts, and for more information on the elements of a font's name, see the entry in this manual's Lexicon for the X utility **xfontsel**.

Colors

X Windows includes tools for describing and managing colors. The monochrome server also manages colors; however, it only recognizes two colors — black and white.

The file **/usr/X11/lib/rgb.txt** defines and names each color that the X server recognizes. The following gives a typical entry:

```
65 105 225     RoyalBlue
```

The three numbers give the intensity settings on the three electron guns in your color monitor: respectively red, green, and blue. Each intensity can range from 0 through 255. Setting all three guns to the maximum intensity, 255, creates white; setting all three to zero creates black.

When you name a color in a defaults file or command-line option, the X server looks up the name in this file and translates it into its numeric settings. If you wish to add a new name for a color, just edit this file. Note that it is unwise to modify or delete any color names from this file; otherwise, if an application requests that color by name, X will not be able to find it.

So far, so good. A problem, however, arises because you can describe up to 16,777,216 colors (that is, 256 cubed), but most VGA cards can display no more than 16 colors at any one time. An application usually requests a set, or *palette*, of colors from the server; the server, then, has to somehow juggle the palettes of all of the applications it is handling into one master palette that it can request from your system's video card.

X manages its palette automatically, and in most instances you do not have to do anything. X Windows for COHERENT does include tools to help you manage colors by hand, should you wish to do so.

If you wish, you can edit **Xconfig** to set the mode of color display. **X** recognizes any of the following settings:

```
DirectColor
TrueColor
PsuedoColor
StaticColor
GrayScale
StaticGray
```

Place the value on a line by itself. The default is **PsuedoColor**.

This concludes our brief introduction to the structure of X. Although X is large and complex, it never departs from the principles outlined here — well, hardly ever. The following section describes how to customize the window manager **twm**; you will see some of these principles in action.

Customing the Window Manager

As noted earlier, the window manager is the master client that manages the appearance of the entire screen. Its window fills the entire screen; thus, the window of the window manager is sometimes called the screen's *root window*.

All other windows on the screen are sub-windows of the root window; thus, the window manager positions all other windows on the screen (or, to be more exact, within the root window), and for supplying services within the root window such as the drop-down menus, the window frames, icons, and window title bars. As a rule of thumb, remember that everything within a window's frame is defined by the client displayed within that window, whereas everything from the frame out is the responsibility of the window manager. You could run more than one application under X without a window manager, but you would have no way to reposition or resize windows, or perform many of the other actions that make X Windows so flexible and useful.

Like many other X clients, the window manager **twm** has a defaults file that defines the resources it uses. We will discuss how to modify this file to change the appearance and behavior of the window manager. First, however, we must introduce the program **xinit**, which invokes the window manager and the other clients that appear on your screen when you invoke X.

Customizing xinitrc

As you recall, to invoke X you must issue the command **startx**. This is actually a shell script that invokes the X utility **xinit**; this utility is responsible for invoking the X server, the window manager, and any other utilities that you want to appear on the screen when you start up X.

When **xinit** is invoked, it reads the file **/usr/X11/lib/xinit/xinitrc**, which gives the default configuration of X. This file reads as follows:

```
xclock -geometry 135x141+15+26 -fg blue -chime -update 1 &
xvt -geometry 80x24+130+146 &
twm
```

The first two commands, **xclock** and **xvt**, invoke the applications **xclock** and **xvt**, and display them on the screen. The argument **-geometry** gives the size of the window and its position on the screen. The first two numbers, which are separated by an 'x', give the size. This can be either in pixels or, as with the **xvt** command, in rows and columns. The second two numbers give, respectively, the X and Y position of the upper left corner of the window. If a number is positive, it is counted from the upper-left corner of the screen; if negative, it is counted from the lower-right corner. Note that both **xclock** and **xvt** are invoked in the background, as shown by the '&' that concludes the command.

The last command, **twm**, invokes the window manager. This is invoked in the foreground, because you want the X server to die when you exit from the window manager. Thus, when you select the command `(Quit_Window_Mgr.)` from the menu **TWM Operations**, the X server also dies, and the console returns to its normal, character-based interface.

Suppose, however, that you want the screen to appear differently. To do so, you would first enter the command

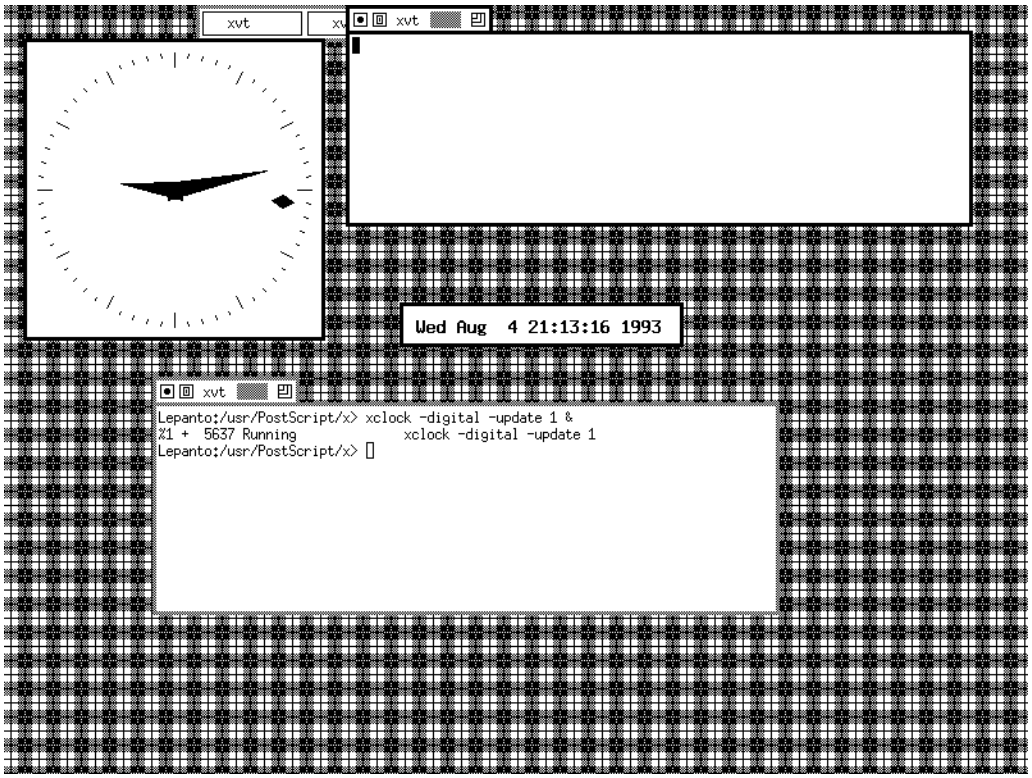
```
cp /usr/X11/lib/xinit/xinitrc $HOME/.xinitrc
```

This gives you a private copy of **xinitrc**. When **xinit** is invoked, it first looks for file **.xinitrc** in your home directory; only if it cannot find this file does it read file **/usr/X11/lib/xinit/xinitrc**.

Now, you can use MicroEMACS or any other text editor to modify your **$HOME/.xinitrc** as you wish. For example, if you wish to tile the root window with a plaid design (so that it resembles a picnic tablecloth), use the command **xsetroot** as follows:

```
xsetroot -bitmap /usr/X11/include/X11/bitmaps/plaid &
```

The next time you invoke X, the root window will appear like this:



For details on the command, **xsetroot** see its entry in this manual's Lexicon.

As another example, suppose that you prefer to have a digital clock rather than an analogue one. Just modify the **xclock** command to the following:

```
xclock -geometry 135x141+15+26 -digital -fg blue -chime -update 1 &
```

You can invoke other X clients, if you wish. For example, many users like to invoke the X client **xbiff**. This client displays a picture of a mailbox; when you receive mail, the flag on the mailbox pops up. Try adding the command

```
xbiff -geometry 91x81+476+44 &
```

after the **xsetroot** command. The mailbox will then be displayed automatically the next time you invoke X.

Many users also like to invoke the X client **xeyes** when they bring up X. This command displays a pair of eyes whose pupils follow the mouse cursor around the screen. Inserting the command

```
xeyes -geometry 150x100+286+66 &
```

into **.xinitrc** brings up the eyes near the top center of your screen.

Note that changing the **-geometry** argument to a command can be rather difficult. The best approach is invoke the client through **xvt**; position and size its window as you want it; then select the `(Info)` option from the menu **WINDOW OPS**, to show you the size and position of the window; and copy this information into **xinit**.

Customizing .twmrc

As shown above, the window manager **twm** is invoked last from within **xinitrc**, so that when it exits the X session will end as well. When **twm** comes up, it reads the file **/usr/X11/lib/twm/system.twmrc**, which sets its default behaviors. **twm** has many features that you can set or invoke, so **twmrc** can be extremely complex. The following introduces a few of the more commonly used features that you can set or invoke from within **twmrc**.

To begin, copy the file **/usr/X11/lib/twm/system.twm** to **$HOME/.twmrc**. **twm** reads **system.twmrc** only if it cannot find **.twmrc** in your home directory. You can now use MicroEMACS or any other text editor to edit your private copy of **.twmrc** to suit your preferences.

.twmrc is divided into three sections:

Variables This section sets switches and variables within **twm**, such as the colors used with various objects, the cursor shapes used in various circumstances, and how icons are managed.

Bindings This binds function keys, mouse buttons, and title buttons to various actions. It also lets you define functions. These functions are much like shell functions, in that they consist of one or more of **twm**'s built-in functions bundled together.

Mouse buttons and function keys can be given different bindings, depending upon the object in which the mouse cursor is positioned. A binding can tell **twm** to display a menu, execute a command under the shell, invoke one of **twm**'s built-in functions, or invoke a user-defined function.

Menus These define the contents of the menus that **twm** displays. Note that the **Bindings** section defines what menus appear when.

The following sub-sections describe these portions of **.twmrc** in more detail. For a full description of **twm**'s variables, functions, and menus, see the entry for **twm** in this manual's Lexicon.

Variables

As noted above, **twm** has many variables that you can set to change its appearance or behavior. Some variables are Boolean and some take a numeric value, but most take a string value as an argument. Some variables also take a *window list*, or a list of windows to which a given argument applies.

As an example of a Boolean variable, consider the variable **DontMoveOff**. When this variable is present in **.twmrc**, the window manager will not let you move a window off the physical screen. Thus, inserting the variable

```
DontMoveOff
```

into your **.twmrc** restricts window movement to the physical screen. You can override this default setting, as will be described.

The variable **NoTitle** is another Boolean variable: if set, **twm** does not display a title bar on its windows. You can, however, use a window list with this variable to restrict it to selected windows. **system.twmrc** defines **NoTitle** as follows:

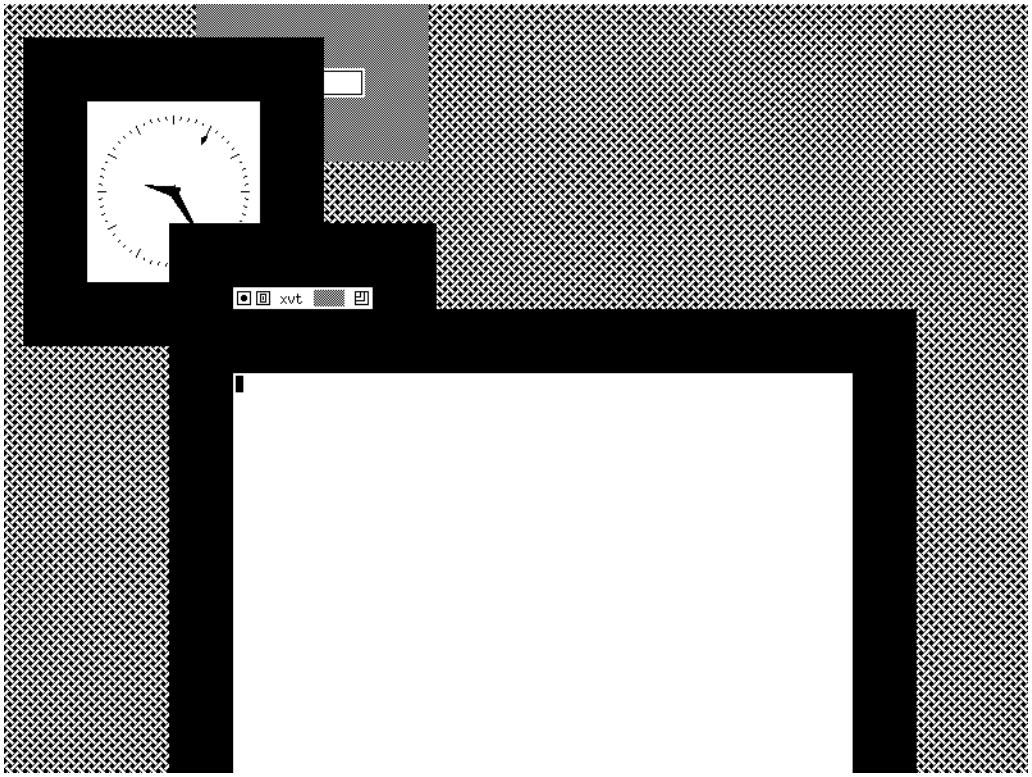
```
NoTitle
{
        "Twm Door"
        "TWM Icon Manager"
        "oclock"
        "xbiff"
        "xclock"
        "xload"
}
```

As you can see, **xclock** is on this list; and when it comes up on your screen, it does not have a title bar.

The variable **BorderWidth** is an example of a variable that takes an integer value. This variable sets the default width, in pixels, of a window's border (or frame). (Note that in X, the terms "border" and "frame" are used almost interchangeably.) The default setting is three. You can change this value. For example, inserting the instruction

```
BorderWidth 50
```

makes the window borders 50 pixels wide; and when your screen comes up, it will look like this:



Fifty pixels probably is too wide a border; but, if you wish, you can thicken the border to four or five pixels to make it easier for you to resize and move the window.

Finally, as an example of a variable that takes one or more string arguments, consider the variable **Cursors**. This variable sets the shape of the cursor in various parts of the screen, or when certain actions are occurring. **system.twmrc** defines this variable as follows:

```
Cursors
{
        Frame           "top_left_arrow"
        Title           "top_left_arrow"
        Icon            "top_left_arrow"
        IconMgr         "top_left_arrow"
        Move            "fleur"
        Resize          "fleur"
        Menu            "sb_left_arrow"
        Button          "hand2"
        Wait            "watch"
        Select          "dot"
        Destroy         "pirate"
}
```

The terms in the left column define a portion of the screen or an action. For the most part, these are self-explanatory: for example, **Move** and **Resize** refer to when you are, respectively, moving and resizing a window; whereas **Frame** refers to when the cursor touches the window border.

TUTORIALS

The right column names the cursor to be used. Because the names are strings, you must enclose them within quotation marks. The names of the cursors are defined in file **/usr/X11/include/X11/cursorfont.h**, with except that the prefix **XC_** is dropped from each name. Read this file for suggestion on other cursors you can use; for example, you can change the **Wait** cursor from **watch** to **coffeemug**, which shows a stylized coffee mug. For a picture of what cursors are available, see the entry for the command **xfd** in this manual's Lexicon.

As noted above, the X cursors are stored in a font; a font, after all, is simply a set of bit-mapped images. A cursor actually consists of two bit-mapped images: one draws the cursor itself, and the second (which is drawn in a constrasting color) forms a mask around the cursor. This is done so that the cursor will never disappear against its background. If you wish, you can use a bit-mapped image of your own creation as a mouse cursor. To do so, simply replace the name of the cursor from the **cursor** font with the name of the bit-mapped image you want, plus its mask. For example, if you replace the definition of the **Frame** cursor with the following:

```
Frame           "/usr/X11/include/X11/bitmaps/star" \
                "/usr/X11/include/X11/bitmaps/starMask"
```

then, the next time you restart X, the cursor will change to an asterisk whenever the mouse cursor touches a window frame.

Some string variables let you set specific values for one or more windows. For example, consider the variable **Color**, which sets the window manager's color scheme under the color server. **system.twmrc** defines it as follows:

```
Color
{
        DefaultBackground               "#5f9ea0"       # cadet blue
        DefaultForeground               "#ffffff"       # white
        BorderColor                     "#00ff00"       # green
        {
        "xload"                         "khaki"
        "xclock"                        "khaki"
        "xbiff"                         "red"
        }
        BorderTileBackground            "black"         # {winList}
        BorderTileForeground            "white"         # {winList}
        MenuBackground                  "firebrick"
        MenuForeground                  "white"
        MenuTitleBackground             "green"
        MenuTitleForeground             "black"
        IconBackground                  "gray85"        # {winList}
        IconForeground                  "brown"         # {winList}
        IconBorderColor                 "yellow"        # {winList}
        IconManagerBackground           "blue"          # {winList}
        IconManagerForeground           "white"         # {winList}
        IconManagerHighlight            "#ffff00"       # {winList}
        MenuShadowColor                 "grey40"
        TitleBackground                 "firebrick3"    # {winList}
        TitleForeground                 "white"         # {winList}
}
```

The left column names the section of the screen being defined. Most of these are self-explanatory.

The right column names the color to be used in that portion of the screen. Because these are strings, you must enclose them within quotation marks. A color can be defined either by name or as a number. The color names that **twm** recognizes are defined in file **/usr/X11/lib/rgb.txt**. A number consists of a pound sign '#' followed by six hexadecimal digits: two each for the red, green, and blue video guns. For example, **#00ff00** defines solid green.

As noted earlier, **Color** (like some other string variables) lets you set colors for individual windows by name. For example, the following sets the colors for the window border:

```
BorderColor                     "#00ff00"       # green
{
"xload"                         "khaki"
"xclock"                        "khaki"
"xbiff"                         "red"
}
```

The default border color is green; however, the windows for the X applications **xload** and **xclock** have a khaki border, and the X application **xbiff** has a red border.

If, for whatever reason, you want **xvt** to have a pink border, insert the line

```
"xvt"                           "pink"
```

into the above list. Then, the next time you invoke X, **xvt** will come up bordered in pink.

For a complete list of the variables that you can set or modify your **.twmrc**, see the entry for **twm** in this manual's Lexicon.

Bindings

The second part of **.twmrc** are bindings: that is, entries that define functions, and entries that bind functions to mouse-button events, function keys, and title-button events.

Function bindings combine one or more existing functions into a new function. For example, the following defines the function **xrdb-color**:

```
Function "xrdb-color"
{
        f.exec "xrdb -remove"
        f.exec "xrdb -DCOLOR -load $HOME/.Xdefaults"
}
```

This function consists of two calls to the function **f.exec**, which is built into **twm**; this function executes its argument under a shell. The two calls to **f.exec** invoke the X utility **xrdb**; the first invocation drops the current resource table, and the second loads the contents of file **$HOME/.Xdefaults** into the server. (For details on what these actions accomplish, see the Lexicon entry for **xrdb**.) The section on menus, below, will show how to call a function. **twm** has a number of functions built into it; for a full list, see the entry for **twm** in this manual's Lexicon.

The binding of a title button consists of the name of the title button, followed by the name of the bit-mapped image to be drawn on the title bar, followed by the action to execute when it is invoked. For example, following entry from **system.twmrc** binds the left-title button:

```
LeftTitleButton         ":menu"     = f.menu "LeftTitleButton"
```

The **LeftTitleButton** is the button that appears immediately to the left of the name of the window in the title bar. **:menu** is the bit-mapped image to display for that button; it is what we have called a stylized 'D'. This menu is built into **twm**. Finally, the binding indicates that when this button is "pressed," **twm** is to invoke its internal function **f.menu** and display the menu called **LeftTitleButton**. This menu, which is described below, is the menu that on the screen has the title **WINDOW OPS**.

A binding for a button or keyboard event consists of three parts: the name of the key or button in question, the context in which the binding applies, and the action to take. The name of the key or mouse button can be combined with metakeys (i.e., the **<shift>** key, the **<ctrl>** key, and the **<esc>** key) to form other key combinations.

Context gives the context in which a pressing a given button or key performs a given action. The following lists the possible contexts:

a	All
f	Frame
i	Icon
m	Icon Manager
r	Root Window
t	Title
w	Window

A button or key can have more than one context; if so, the context letters must be separated by a vertical bar '|'. Note that with regard to the **w** context, this simply sets a default action; the application, of course, governs what goes on within its own window, and can overrule this setting should it wish.

The action calls a function, as shown above.

The following gives the bindings from **system.default** for mouse button 1 (that is, the left-mouse button):

```
Button1 = c|m          : i|f|t|w              : f.raise
Button1 =              : root                 : f.menu "Applications"
Button1 =              : t|i                  : f.move
Button1 =              : frame                : f.resize
Button1 =              : m|i                  : f.iconify
```

The following discusses these lines in order:

1. This first line defines Button1 with a **<ctrl>** or **<esc>** (meta) key. If the mouse cursor is in an icon, on the window frame, on the title bar, or within a window, then holding down the **<ctrl>** key and pressing the left-mouse button invokes the **twm** function **f.raise**, which raises the window to the foreground.
2. The second and succeeding lines contain no key entries, which means that this button is not used with metakeys. The second line instructs **twm** to invoke the function **f.menu** to display the menu **Applications** should you press the left-mouse button while the mouse cursor is positioned over the background of the root window.
3. Invoke the **twm** function **f.move**, which moves the current window, should you press the left-mouse button while the mouse cursor is positioned over the title bar or the window's icon.
4. Invoke the **twm** function **f.resize**, which resizes a window, should you press the left-mouse button while the mouse cursor is positioned over the window's frame.
5. Finally, invoke the **twm** function **f.iconify**, which iconifies a window, should you press the left-mouse button while the mouse cursor is positioned over the icon manager or the window's icon. Note that this contradicts the definition given in line 3, which told **twm** to raise the window should you click the left-mouse button while the cursor is positioned over the window's icon. In the case of contradiction like this, **twm** uses the last binding to appear in its resource file.

You can modify these bindings if you wish. For example, you can modify the setting for the left-mouse button in the context of the window's frame to the following:

```
Button1 =      : frame : f.destroy
```

Normally, when you click the left-mouse button while the mouse cursor is resting on the window's frame, **twm** resizes the window; this change destroys the application and closes its window. This is an extreme case, but it shows that you can rebind the buttons and keys to suit your preferences in almost every case.

For a complete list of the elements that you can bind, and the internal **twm** functions to which you can bind them, see the Lexicon entry for **twm**.

Menus

Menus are in many ways the easiest part to define in a the **twm** resource file. A menu begins with the keyword **Menu**, followed by the menu's name. Then comes the default color scheme for the menu, foreground first and then background. The entries for the menu follow, between braces. Each entry consists of the stub that appears in the menu, followed by an optional color scheme, then the action to take when this entry is selected.

The following gives the definition of the menu **Applications** from **system.twmrc**:

```
Menu "Applications" ("black":"lightseagreen")
{
        "APPLICATIONS" ("black":"lightseagreen")f.title
        "Font Select" ("lightseagreen":"black")!"xfontsel &"
        "Puzzle"                              !"puzzle &"
        "XBiff"                               !"xbiff &"
        "XCalc"                               f.menu " Calculator "
        "XClock"                              !"xclock -chime -fg blue -update 1 &"
        "Xeyes"                               !"xeyes -fg red &"
        "XLoad"                               !"xload &"
        "XLogo"                               !"xlogo &"
        "XMag"                                !"xmag &"
        "XTerm"                               !"xvt -ls -cr red &"
        "XTetris" ("lightseagreen":"black")   !"xtetris &"
}
```

This menu is shown in one of the figures reproduced earlier. Most of the entries are self-explanatory. Note that the '!' is a synonym for **f.exec**, which executes its argument under the shell.

The entry **APPLICATIONS** invokes the **twm** function **f.title**, which indicates that this is the title entry for the menu. You cannot select it.

The entry for **XCalc**

```
"XCalc" f.menu " Calculator "
```

invokes the function **f.menu**. This function displays a button at the right of the menu entry, and displays a sub-menu when the button is touched. Its argument is the name of the menu to drop down, in this case the menu entitled **Calculator**, which is also defined in **system.twmrc**.

If you wish, you can add commands to this menu: just insert a line in the appropriate spot. The entries in the menu are sorted by hand; **twm** does not rearrange their order in any way. For example, to insert into this menu the game **xgas**, shown above, just add the line

```
"Xgas"  !"xgas -fg red &"
```

directly after the entry labelled **Xeyes**.

If you wish, you can also change the arguments to existing commands. For example, the entry

```
"XClock"        !"xclock -chime -fg blue -update 1 &"
```

invokes the X client **xclock** invokes an analogue clock with a blue face, which is updated once every second, and chimes on the hour and half-hour. If you wish to eliminate chiming and have a digital clock instead, modify this command to read:

```
"XClock"        !"xclock -digital -fg blue -update 1 &"
```

For the options available with each of these commands, see its entry in this manual's Lexicon.

The color settings are used only by the color server. If you wish, you can give a separate color combination to each entry in the menu. Note that the first and last selectable entries in this menu both have the color settings of:

```
("lightseagreen":"black")
```

There is a reason for giving the same colors to the first and last entries: if the first and last color settings in the menu are not exactly the same, **twm** generates colors for the menu that fade from the first known pair of colors to the last known pair. To see this in action, try removing the first instance of:

```
("lightseagreen":"black")
```

If a menu does not set any colors, the server uses the settings for **MenuBackground** and **MenuForeground** in the variable **Color**. The monochrome server ignores all color settings, and uses the settings for **MenuBackground** and **MenuForeground** in the variable **Monochrome**.

This concludes the discussion of how to customize the window manager. For more information, see the Lexicon entries for **twm**, **xinit**, and **xsetroot**.

Where To Go From Here

This concludes our introduction to the X Window System. For more information, see the following chapter, which introduces the clients and utilities that comes with X Windows for COHERENT. The Lexicon discusses each client and utility in detail. If you wish to explore X further, we suggest that you buy the books named in the introduction to this manual.

However, there is no better teacher than experience. We suggest that you work with X, try modifying the defaults files (or, to be exact, *copies* of the defaults files), and play with the system. X Windows is well designed and robust. You will become proficient quickly. We hope that you enjoy using X Windows!

X Windows Clients

This chapter introduces the clients and utilities included with X Windows for COHERENT.

Strictly speaking, a *client* is a program that registers with, and runs under, the X server. "Client" often is used as a synonym for the term *application*. This manual uses the term *utility* to refer to a program that helps you to manage and run the X Window System itself; and the term *client* to refer to a program with which you can perform some task under the X Window System (e.g., edit a file or play a game).

This chapter introduces the utilities and clients included with X Windows for COHERENT. It then gives examples of how to modify an application's appearance or behavior by changing its resources.

X Utilities

A utility helps you to run the X Window System itself. X Windows for COHERENT includes utilities to help run every aspect of the X system. The following introduces them by category.

Bit Maps

A *bit map* is an image that is composed of black and white pixels within a defined space. X uses bit maps extensively; for example, the shapes of the mouse cursor, fonts of alphabetic characters, buttons, and icons are all bit maps.

X stores a bit map in the form of a array. For example, the file **/usr/X11/include/X11/bitmaps/xlogo32**, which is a bit map of the X logo, consists of the following:

```
#define xlogo32_width 32
#define xlogo32_height 32
static char xlogo32_bits[] = {
   0xff, 0x00, 0x00, 0xc0, 0xfe, 0x01, 0x00, 0xc0, 0xfc, 0x03, 0x00, 0x60,
   0xf8, 0x07, 0x00, 0x30, 0xf8, 0x07, 0x00, 0x18, 0xf0, 0x0f, 0x00, 0x0c,
   0xe0, 0x1f, 0x00, 0x06, 0xc0, 0x3f, 0x00, 0x06, 0xc0, 0x3f, 0x00, 0x03,
   0x80, 0x7f, 0x80, 0x01, 0x00, 0xff, 0xc0, 0x00, 0x00, 0xfe, 0x61, 0x00,
   0x00, 0xfe, 0x31, 0x00, 0x00, 0xfc, 0x33, 0x00, 0x00, 0xf8, 0x1b, 0x00,
   0x00, 0xf0, 0x0d, 0x00, 0x00, 0xf0, 0x0e, 0x00, 0x00, 0x60, 0x1f, 0x00,
   0x00, 0xb0, 0x3f, 0x00, 0x00, 0x98, 0x7f, 0x00, 0x00, 0x98, 0x7f, 0x00,
   0x00, 0x0c, 0xff, 0x00, 0x00, 0x06, 0xfe, 0x01, 0x00, 0x03, 0xfc, 0x03,
   0x80, 0x01, 0xfc, 0x03, 0xc0, 0x00, 0xf8, 0x07, 0xc0, 0x00, 0xf0, 0x0f,
   0x60, 0x00, 0xe0, 0x1f, 0x30, 0x00, 0xe0, 0x1f, 0x18, 0x00, 0xc0, 0x3f,
   0x0c, 0x00, 0x80, 0x7f, 0x06, 0x00, 0x00, 0xff};
```

The manifest constants **xlogo32_width** and **xlogo32_height** give the width and height of the image, in pixels. The zero-bits within the array represent white pixels, whereas the non-zero bits represent black pixels.

If you wish, you can draw new bit maps or edit existing bitmaps. X Windows for COHERENT includes the following tools for working with bit maps:

bitmap	Bit map editor
atobm	Convert ASCII to an X bit-mapped image
bmtoa	Convert an X bit-mapped image to ASCII

bitmap is a bit-map editor. With it you can draw a bit-mapped image using the mouse. It also has tools for copying or inverting regions of the bit map, drawing geometric shapes, rotating bit maps, and the like.

bmtoa converts a bit map to an ASCII format that you can edit with an ordinary text editor. For example, the command

```
bmtoa /usr/X11/include/X11/bitmaps/xlogo32
```

turns the bit map shown above into the following:

```
########----------------------##
-########----------------------##
--########--------------------##-
---########------------------##--
---########----------------##---
----########--------------##----
-----########------------##-----
------########----------##------
------########----------##------
-------########--------##-------
--------########------##--------
---------########----##---------
---------########---##----------
----------########--##----------
-----------#######-##-----------
-----------#####-##------------
------------####-###------------
-------------##-#####-----------
------------##-#######----------
-----------##--########---------
-----------##--########---------
----------##----########--------
---------##------########-------
--------##--------########------
-------##---------########------
------##-----------########-----
------##------------########----
-----##--------------########---
----##---------------########---
---##-----------------########--
--##-------------------########-
-##---------------------########
```

By default, **bmtoa** represents a white pixel with a hyphen '-' and a black pixel with a pound sign '#'.

Finally, **atobm** turns an ASCII image into a bit map that X can use. You could, for example, use a text editor to edit the image generated by **bmtoa**, then use **atobm** to re-compile the image into bit-map format.

As a side note, you can use a bit map to "tile" the root window of your screen — simply invoke the command **xsetroot** with its option **bitmap**. For example, the command

```
xsetroot -bitmap /usr/X11/include/X11/bitmaps/xlogo32
```

tiles the root window with the X logo.

Colors

X Windows for COHERENT includes the following utilities for manipulating colors:

showrgb	Un-compile an RGB color-name data base
xcmsdb	Manipulate xlib screen-color characterization data
xstdcmap	X standard color-map utility

showrgb names the colors that the server recognizes. The file **/usr/X11/lib/rgb.txt** gives an example of its output, and also shows colors that the X color server recognizes by default.

xcmsdb and **xstdcmap** let you display and manipulate color information within widgets and the server. Using these utilities requires a detailed knowledge of X internals. For details, see the entry in the Lexicon for each utility.

Fonts

A *font* is a set of bit-mapped images that form the letters of the alphabet and commonly used punctuation marks. X Windows for COHERENT includes the following utilities for manipulating text fonts:

bdftopcf	Generate a PCF font from a BDF file
mkfontdir	Create file fonts.dir from directory of font files
xfd	Display all the characters in an X font
xfontsel	Interactively select X11 fonts
xlsfonts	List fonts being used on a server

TUTORIALS

X reads fonts from the directories named in its **FontPath**, which is set in the file **/usr/X11/lib/Xconfig**. Each directory in the **FontPath** contains a file named **fonts.dir**, which gives the full, 14-part name of each font in the directory and the file that holds it. This is done because font files normally are compressed, and uncompressing and searching all of the files in a directory to see if a given font was available would be unacceptably time-consuming. Thus, when you copy a new font into a font directory, you must modify **fonts.dir**, or X will not be able to find the font. The utility **mkfontdir** reads the fonts in a given font directory and rebuilds **fonts.dir** automatically.

The utility **xlsfonts** displays the full, 14-part of each font that is currently available to the X server.

The utility **xfontsel** lets you select a font interactively. It uses a series of 14 drop-down menus to help you build a font name, based on the fonts that are available to your system. The Lexicon entry for this utility also explains just what the elements of a font name mean.

Fonts can be encoded in a number of different ways. X Windows for COHERENT uses fonts that are in the portable compiled format (PCF). Fonts, however, often are shipped in the bitmap distribution format (BDF). The utility **bdftopcf** converts a BDF font into PCF so you can use it on your system.

Finally, the utility **xfd** displays a font. For example, the command

```
xfd -fd 6x10 -center
```

displays the font kept in file **/usr/X11/lib/fonts/misc/6x10.pcf.Z**. When you click on a cell of the display, **xfd** displays detailed information about the character in that cell. For an example of a displayed font (albeit a special font), see the entry for **xfd** in this manual's Lexicon.

Manipulating the Console

The following utilities let you modify the console's appearance:

xrefresh	Refresh all or part of an X screen
xset	Set preferences for the display
xsetroot	Set preferences for the root window

xrefresh redraws a given window or the entire screen, whichever you prefer. You can use this to redraw a window or the screen, should it become cluttered with stuff from other, non-X processes, or should it somehow become confused.

xsetroot lets you modify the appearance of the root window — that is, the window that forms the background of the screen. You can change the window to a solid color, a pair of stippled colors, a gray scale, or tile it with a bit-mapped image. It also lets you change the mouse cursor that is displayed against the root window.

Finally, the utility **xset** lets you set parameters for the display, and for other input devices. For example, **xset** lets you set the acceleration rate for the mouse, and the loudness of both the key click and the bell. To set the key click to 50% of its maximum volume, type:

```
xset c 50
```

Note that not every computer lets you reset the volume of the keyclick or bell. For details, see the Lexicon entry for **xset**.

Programming Tools

X Windows for COHERENT includes the following tools for remaking X utilities:

imake	C preprocessor interface to the **make** utility
makedepend	Create dependencies in **makefile**s
mkdirhier	Make a directory hierarchy
xmkmf	Create a **Makefile** from an **Imakefile**

None of these tools interact with the X server; if you wish, you can use them with your ordinary, character-based applications as well as on X applications.

imake is a superset of the COHERENT utility **make**. It understands a more complex set of dependencies than those encompassed by **make**'s grammar. These dependencies include widgets, operating system, and microprocessor. **imake** reads an **Imakefile**, which contains dependencies plus C-preprocessor directives. You can define constants on the **imake** command line to determine what the resulting program will look like. Note that you will seldom, if ever, need to invoke **imake** directly. Usually, you will invoke it via the script **xmkmf**, which is described below.

makedepend reads a set of C source files and header files, and builds a table of dependencies from them. From this table, you can construct a **makefile** or an **Imakefile**.

mkdirhier creates a directory hierarchy. That is, if the parent directories of a target directory do not exist, **mkdirhier** creates them first, then creates the directory you requested.

Finally, **xmkmf** is a script that invokes **imake** to build **Makefile** from an **Imakefile**. It passes appropriate arguments to **imake** to ensure that it builds a **Makefile** that runs correctly on your system.

These utilities are discussed at greater length in the next chapter.

Resources and Properties

The following utilities help you examine and manage resources and properties:

appres	List an application's resource data base
editres	Resource editor for X Toolkit applications
listres	List resources in widgets
viewres	Graphical class browser for **Xt**
xprop	Display the X server's properties
xrdb	Read/set the X server's resource data base

appres prints on the standard output the resources that an application uses. The following gives a portion of the output of the command **appres XTerm**:

```
*VT100*color2: green
*VT100*font5:  9x15
*VT100*color6: cyan
        ...
*VT100*font1:  nil2
*VT100*font4:  7x13
*VT100*color5: magenta
*tekMenu*vtshow*Label: Show VT Window
        ...
*tekMenu*tektext3*Label:      #3 Size Characters
*tekMenu.Label:       Tek Options
*fontMenu*font5*Label: Large
*fontMenu*font6*Label: Huge
*fontMenu*font2*Label: Tiny
        ...
*mainMenu.Label:      Main Options
*vtMenu*hardreset*Label:      Do Full Reset
*vtMenu*scrollbar*Label:      Enable Scrollbar
*vtMenu*scrollkey*Label:      Scroll to Bottom on Key Press
*vtMenu*scrollttyoutput*Label:       Scroll to Bottom on Tty Output
        ...
*vtMenu*autolinefeed*Label:   Enable Auto Linefeed
*vtMenu*altscreen*Label:      Show Alternate Screen
*vtMenu*appcursor*Label:      Enable Application Cursor Keys
*vtMenu*softreset*Label:      Do Soft Reset
*vtMenu*appkeypad*Label:      Enable Application Keypad
        ...
*tek4014*fontSmall:   6x10
```

Resources are discussed in more detail below.

editres is an interactive program that lets you edit an application's resources on the fly. By selecting options from buttons and menus, you can add or delete resources from an application, or change a resource's value, then view the result and dump the altered resources into a file. In effect, **editres** gives you a way to edit an application's resource file interactively. Playing with **editres** is a good way to learn about resources.

listres lists resources used in a widget. For example

```
listres -all
```

prints information about all known widgets.

In the context of X, the term *property* means a string that holds information about an application, such as its color or the size of its window. Properties are stored within the server, so they can be read by all other clients, including the window manager. **xprop** displays the properties associated with a given client. The following gives the output

of **xprop** when invoked for the X client **xclock**:

```
WM_STATE(WM_STATE):
                window state: Normal
                icon window: 0x0
WM_PROTOCOLS(ATOM): protocols  WM_DELETE_WINDOW
WM_CLASS(STRING) = "xclock", "XClock"
WM_HINTS(WM_HINTS):
                Client accepts input or input focus: False
                Initial state is Normal State.
                bitmap id # to use for icon: 0x1000001
                bitmap id # of mask for icon: 0x1000003
WM_NORMAL_HINTS(WM_SIZE_HINTS):
                user specified location: 15, 26
                user specified size: 135 by 141
                window gravity: NorthWest
WM_CLIENT_MACHINE(STRING) = "chelm"
WM_COMMAND(STRING) = { "xclock", "-geometry", "135x141+15+26", \
                "-fg", "blue", "-chime", "-update", "1" }
WM_ICON_NAME(STRING) = "xclock"
WM_NAME(STRING) = "xclock"
```

Finally, the utility **xrdb** reads and sets the X server's resource data base. It manipulates the contents of the properties **RESOURCE_MANAGER** and **SCREEN_RESOURCES**. Applications read these properties to obtain resources that are common to all resources under a given server. Note that the contents of these properties usually override what an application may read from its defaults file. These properties are also read by applications that do not normally read a defaults file (e.g., **xbiff**), and so can be used to modify them.

xrdb is of particular use in a networked environment. It lets you embed resources within your machine's server, so that every client that appears on your machine, regardless of the machine it originates from, conforms to the way you want it to appear. This spares you from having to store a defaults file on every machine from which you might invoke a client.

xrdb is invoked by default by the X display manager **xdm**, which is not included with X Windows for COHERENT. If you wish to set defaults with **xrdb**, you should call it from within the file **$HOME/.xinitrc**. The next sections gives some examples of how to use **xrdb** to set resources.

System Monitoring

The following utilities help you keep an eye on your system:

xauth	Display/edit authorization information
xdpyinfo	Display information about an X server
xev	Print contents of X events
xlsatoms	List atoms defined on server
xlsclients	List client applications running on a display
xwininfo	Display information about a window

xauth lets you create or edit an authorization file. This file determines who from what system can execute what clients on your system. This utility is used mainly in networked environments, where the question of system security becomes very important.

xdpyinfo displays information about the X server running on your system. The following gives a portion of its output:

```
name of display:    :0.0
version number:    11.0
vendor string:    Ready-to-Run Software, Inc.
vendor release number:    5000
maximum request size:  262140 bytes
motion buffer size:  0
bitmap unit, bit order, padding:    8, MSBFirst, 32
```

If your server were handling more than one display, information would be shown for each.

An *event* is something that occurs on your system that sends a signal to the X server. For example, the mouse sliding on your desk is an event, because the server must reposition the mouse cursor. Events are generated every time you press a mouse button, press a key on the keyboard, or in any other way interact with your system. The X utility **xev** displays a small window on the screen, and then displays information about every events received by

the server.

In the context of X, an *atom* is an elemental portion of the server that is available to clients. For example, properties are atoms, as are the names of fonts, some static strings, and other information. The X utility **xlsatoms** lists your server's atoms. The following gives a portion of its output:

```
1       PRIMARY
2       SECONDARY
3       ARC
4       ATOM
        ...
9       CUT_BUFFER0
        ...
126     -Misc-Fixed-Medium-R-Normal--10-100-75-75-C-60-ISO8859-1
127     EditresComm
        ...
134     -Misc-Fixed-Bold-R-Normal--13-120-75-75-C-80-ISO8859-1
```

The utility **xlsclients** displays information about all of the clients currently running under your server.

Finally, **xwininfo** displays information about a window. The following gives a sample of its output:

```
xwininfo: Window id: 0x1000009 "xclock"

  Absolute upper-left X:  18
  Absolute upper-left Y:  29
  Relative upper-left X:  0
  Relative upper-left Y:  0
  Width: 135
  Height: 141
  Depth: 1
  Visual Class: StaticGray
  Border width: 0
  Class: InputOutput
        ...
  Corners:  +18+29  -647+29  -647-430  +18-430
  -geometry 135x141+15+26
```

Miscellaneous Utilities

The following utilities do not fit neatly into any of the above categories. These miscellaneous utilities, however, include some of the most useful and interesting programs shipped with X Windows for COHERENT:

resize	Set environmental variables to show window size
sessreg	Manage utmp/wtmp entries for non-init clients
startx	Initiate an X session
twm	Tab Window Manager for the X Window System
xclipboard	Hold multiple selections for later retrieval
xcmstest	XCMS test program
xcutsel	Copy text between the cut buffer and the primary selection
xinit	Initiate the X Window System
xkill	Kill an X client
xmodmap	Modify X keymaps

startx, **twm**, and **xinit** were introduced in the previous chapter.

The utility **resize** reads the size of the current **xvt** window, then prints onto the standard output a shell script that, when run, sets the environmental variables **ROWS** and **COLUMNS** to reflect the size of the window. These variables can be read by programs that run within that window, such as screen editors, so they can size themselves properly. Note that most COHERENT programs, such as MicroEMACS and **vi**, cannot yet resize themselves.

sessreg assists with logging within the system file **/etc/utmp** all X clients that run under the X server. It does nothing unless you have enabled process logging.

X includes a system-wide facility for cutting and pasting text. With this, you can cut text from one window and paste it into another. Normally, X has only one buffer into which you can store cut text. The utility **xclipboard** stores an indefinite number of text "cuttings", and retrieve them for repasting an indefinite number of times. For more information on cutting and pasting, see the entry for **xclipboard** in this manual's Lexicon.

TUTORIALS

Earlier releases of X did not use the property **PRIMARY** to store cut text; rather, they stored cuttings only in a cut buffer. If you cut and paste text between an up-to-date X client and an older one, you may find that the older one does not reset the property **PRIMARY** correctly, and thus does not produce what you think it will when you cut text under it. **xcutsel** copies text between a cut buffer and the property **PRIMARY**, to help keep different generations of applications synchronized. All of the utilities and clients included with X Windows for COHERENT support the latest implementation of X; therefore, you should not need this utility unless you import an obsolete application from elsewhere.

xkill kills an X program. Note that a killed program often leaves debris in memory and on the file system, so you should use **xkill** only in the direst extremity.

The X server has its own internal keyboard mapping. For most users, this does not create a problem, because both they and the X server use the default U.S. keyboard mapping; however, if you have used the COHERENT driver **nkb** to load a foreign keyboard or to customize a keyboard to your preferences, this behavior of the server's can create serious difficulty. The utility **xmodmap** lets you modify the mappings that the X server recognizes for the keyboard and the mouse. With this program, you can (for example) exchange the left mouse button with the right mouse button, switch the **<ctrl>** key with the **<CapsLock>** key, and perform other tasks to help your system work as you prefer. The file **/usr/X11/lib/.Xmodmap.ger** gives an example script that remaps the keyboard for X; in this case, it remaps the keyboard to the German standard. For details, see the entry for **xmodmap** in this manual's Lexicon.

This concludes our introduction of the utilities included with X Windows for COHERENT. The next section introduces clients.

Clients

X Windows for COHERENT includes a selection of *clients* — that is, programs that run under the X server and let you do something that is not necessarily related to the running of X itself. It is in the wealth of clients available for it that the true power, and usefulness, of X becomes apparent.

Games

The following are just for fun:

ico	Animate an icosahedron or other polyhedron
maze	Create and solve a random maze
puzzle	The X scrambled-number game
xeyes	Display two roving eyes
xgas	Animated simulation of an ideal gas
xtetris	Wildly amusing implementation of Tetris

ico draws a polyhedron, and bounces it around the screen. The object can be either a wire-frame outline, or solid. Note that unless you have a very robust system, the animation will be rather jerky, and **ico** will "suck up" practically all of your system's computation cycles.

maze draws, and then solves, a random maze. You cannot play this game interactively, but it does appear exciting on the screen.

puzzle implements a scrambled-tile game. It displays a window divided into 16 cells. Fifteen of the cells contain numbered tiles, the 16th is empty. Clicking one button scrambles the tiles. When you click a tile, it (or its row or column) slides into the empty cell, if possible; by maneuvering the tiles, you can un-scramble them. When you give up, you can click another button and have **puzzle** un-scramble itself.

xeyes displays a pair of "eyes" on the screen. The pupils of the "eyes" move to follow the mouse cursor around the screen.

xgas models the random motion of gas molecules in a heated, divided chamber. By setting command-line options, you can set parameters of the molecules' movement, such as the degree of randomness with which they bounce off the chamber's walls. By dragging sliders, you can change the temperature of either of the two sides of the chamber.

Finally, **xtetris** is a implements the popular game Tetris.

Observing the System

In addition to the utilities that help you monitor the operation of X itself, X Windows for COHERENT also includes two clients to help you observe your COHERENT system:

xbiff	Notify the user that mail has arrived
xload	Display your system's load average

xbiff displays a bit map of an old-fashioned mailbox. When you receive mail, the flag on the mailbox pops up.

xload displays a histogram — that is, a bar graph — that shows the load on your system. Every few seconds, it measures activity on your system and adds a new bar to the graph. You can use client to measure roughly how much in the way of system resources a given program consumes.

Pretty Pictures

The following clients show some of the graphics capabilities of X:

xlogo	Display the X Window System logo
xmag	Magnify a part of the screen
xgc	X graphics demonstration

xlogo simply displays the X logo in a window. This is not a bit-mapped image, because when you resize the window, the X logo changes size to match it.

xmag magnifies part of the screen. It translates each pixel of the magnified portion of the screen into a cell in a grid. With **xmag**, you can see exactly how an image is built of a pixel map.

Finally, **xgc** demonstrates X graphics. It displays a screen with a great number of buttons and sliders on it. By pressing buttons, you can construct images and play with X's graphics capabilities. Playing with **xgc** is a good way to learn about X graphics.

Timepieces

X helps you keep track of the time:

oclock	Display an analogue clock
xclock	Display a clock

Both of these clients display a clock on the screen, which displays the time as your system understands it. These clients differ mainly in the shape of their windows, and in the fact that **xclock** offers some extra features — it can display a digital clock, and "chime" on the hour and half-hour, if you wish. Most users permanently display either **oclock** or **xclock** on their X screen.

Tools

Finally, the following clients do not fall into any of the above categories. These include some of the most useful and interesting of the X programs included with X Windows for COHERENT:

xcalc	Scientific calculator for X
xedit	Simple text editor for X
xpr	Print a dump of an X window
xterm	Terminal emulator for X
xvt	VT100 emulator
xwd	Dump an image of an X window
xwud	Un-dump a window image

xcalc displays a picture of a scientific calculator, either a Texas Instruments 30 or a Hewlett-Packard 10C, whichever you prefer. You can use the mouse to press the buttons on the calculator and so perform computations, just as with a real calculator. The virtual calculator implements most of the features of a real scientific calculator — although clicking virtual buttons with a mouse is more difficult than pressing real buttons with your fingers.

xedit is a simple text editor for X. Its default keystrokes closely resemble those used by MicroEMACS, with the exception that **xedit** supports only one window and buffer at a time. (Of course, under X this is not much of a restriction, because you can invoke multiple **xedit** sessions and cut-and-paste among the windows.)

xvt emulated a DEC VT-100 terminal. It opens a window, logs into your system from it via a pseudo-tty, while emluating a VT-100 terminal. Normally, you will run a shell in this window, although you can invoke **xvt** to run a COHERENT program (such as an editor or a data-entry program) instead of shell — just as if you were logging in

from another terminal. On a networked X system, you can have, on one screen, multiple **xvt** windows, each logged into a different system. You can also use X's cut-and-paste facility to cut text from one terminal window and paste it into another.

xterm is an expanded — and more robust — version of **xvt**. It emulates a Tektronix terminal as well as a VT-100. It also includes a number of features that let you set colors and features more easily. Note that unlike **xvt**, **xterm** emulates the VT-100's graphics characters; thus, you can display such COHERENT programs as **vsh** and have them appear the same (or almost the same) as they do when shown through the ordinary, non-X console interface.

xwd dumps an image of a window into a file. You can select a window by name, or dump the root window — which, in effect, saves an image of the entire screen (including menus). The program **xwud** un-dumps a dumped image, by displaying it in a window.

xpr prints an image dumped by **xwd**. By default, it generates PostScript, although you can instruct it to generate code for a variety of other printers as well.

For what it's worth, the images in this manual were captured with **xwd**, then post-processed with **xpr**. The PostScript output of **xpr** was edited slightly by hand, then patched into the manual's **troff** sources by using a set of specially written **troff** macros.

Customizing X Programs

As noted in the previous chapter, you can customize an X application through any of three ways: (1) by setting command-line options, (2) by modifying its defaults file, or (3) by modifying the X server's resource data base.

The Lexicon entry for an application describes the command-line options that are available with that application. These work in exactly the same way as with any other COHERENT application, and are largely self-explanatory. Methods 2 and 3, however, depend upon setting *resources*, which can be rather tricky.

Resources

As explained in the previous chapter, most X applications are constructed in whole or in part from *widgets*. A widget bundles a a graphical image with a routine that invokes an action when a selected event occur. For example, a widget may dictate that when a button (the graphical image) is clicked (the event), a menu appears (the invoked action).

Widgets often are built out of other widgets. A widget that comprises part of one or more other widgets is called a *widget class*.

A *resource* is an aspect of a widget, such as its color, size, or shape. The syntax of a resource string mirrors the structure of a widget, as follows:

[*app***]*|.[***class***]*|.[...*|. ...]]*|.[***resource***]:***value***

app names the application in question. If no application is named and the resource is in the X server's resource data base, then it applies to all applications. If, however, no application is named and the resource is in an application's defaults file, the resource applies only to the application in question.

class names a widget class. A widget class can itself be built out of other widget classes, so a resource string can be an indefinite number of classes.

resource names the particular resource being set.

Finally, *value* gives the value to which you are setting *resource*. This can be a Boolean setting (**True** or **False**), a number, or a string, depending on the aspect being modified.

The elements of a resource are linked by either a period '.' or an asterisk '*'. A period binds tightly: that is, no widget classes can intervene between two classes named in the resource. An asterisk binds loosely: that is, an indefinite number of widget classes can come between the two widget classes so named.

For example, the following gives two lines from the file **/usr/X11/lib/app-defaults/XCalc**:

```
*Font:                  -*-helvetica-medium-r-normal--*-100-*-*-*-*-*-*
*bevel.screen.LCD.Font: -*-helvetica-bold-r-normal--*-120-*-*-*-*-*-*
```

The first line sets the **Font** for every widget (as indicated by the single preceding asterisk) to ten-point Helvetica medium. The second line overrides this default to set the **Font** within the widget **bevel.screen.LCD** (which is the liquid-crystal display within calculator's "screen") to 12-point Helvetica bold. A **Font** widget naturally must be set to a string.

The following gives two more resources from **XCalc**:

```
*ti.bevel.screen.LCD.width:   108
*hp.bevel.screen.LCD.width:   180
```

The first line sets the width of the virtual liquid-crystal display for the Texas Instruments calculator; the second gives the same for the Hewlett-Packard calculator. Here, the use of the period to bind tightly the classes of widget ensures that the dimensions are exactly on the correct virtual calculator. As you can see, a **width** resource requires a dimension, usually pixels.

Modifying Applications

Before we begin, the following examples involve editing files that define how X functions. Before you edit any file, *make a backup copy!* This will let you back out of any *cul-de-sac* you may get yourself into through error or mishap.

To begin, you will recall that the client **xbiff** displays on the screen a small window that contains a bit map of an old-fashioned mailbox. When new mail arrives, the bit map changes to one with the flag popped up that is displayed in reverse video.

X Windows for COHERENT comes with many bit-mapped images that you can use with existing applications. Two, named **mailfull** and **mailempty**, respectively show a full and empty mail in-tray, much like you may have on your desk. Suppose, for the sake of argument, that you wanted to use these bit-mapped images in place of the default mailboxes. You can do this by resetting the appropriate resources, and commanding **xbiff** to use them instead of its built-in defaults.

The first step is to check the Lexicon entry for **xbiff**. Among other things, this names the resources that **xbiff** uses. This entry shows, among many others, the following resources:

fullPixmap(class **Pixmap)**
Name the bit map to display when mail arrives.

fullPixmapMask(class **PixmapMask)**
Name the mask for the bit map to display when mail arrives.

emptyPixmap(class **Pixmap)**
Name the bit map to display when no new mail is present.

emptyPixmapMask(class **PixmapMask)**
Name the mask for the bit map to display when no new mail is present.

These look like the ones we need to modify. The first part of each entry names the widget; its class is given in parentheses.

The next step is to write the resources that we want to use. These are as follows:

```
xbiff*fullPixmap:mailfull
xbiff*fullPixmapMask:mailfullmsk
xbiff*emptyPixmap:mailempty
xbiff*emptyPixmapMask:mailemptymsk
```

The prefix **xbiff** indicates that these settings are to apply only to this application. The asterisk '*' means that an indefinite number of widget classes can occur between the name of the application and the widget in which we are interested; this spares us the trouble of having to build the entire widget tree — and having to rebuild the tree should the designers of **xbiff** decide in the future to insert another layer or two into the widget hierarchy. Finally, value names the file in directory **/usr/X11/include/X11/bitmaps** that holds the bit map we want.

xbiff does not read a defaults file out of **/usr/X11/lib/app-defaults**; but we can still make **xbiff** use our new resources by using the X utility **xrdb** to load them into the X server's resources data base. We save these resources into a file — we will use the conventional file **$HOME/.Xdefaults**, although the name of this file doesn't really matter — then type the following command:

```
xrdb -merge < $HOME/.Xdefaults
```

This command merges the contents of **$HOME/.Xdefaults** into the X server's resource data base. So, the next time you invoke **xbiff**, you see:

instead of the old-fashioned mailbox.

To extend this example, the documentation for **xbiff** also mentions the following two resources:

shapeWindow(class **ShapeWindow)**
Specify whether to shape the window to the **fullPixmapMask** and **emptyPixmapMask**. The default is false.

flip(class **Flip)**
Invert the image when new mail arrives. The default is true.

To change these defaults, insert the following lines into **$HOME/.Xdefaults**:

```
xbiff*shapeWindow:True
xbiff*flip:False
```

Then, type:

```
xrdb -merge < $HOME/.Xdefaults
```

The next time you invoke **xbiff**, its window will be shaped that of the bit-mapped mask; and it will not pop into reverse video when mail arrives.

If an application uses a defaults file, you can simply edit that file to change a resource. For example, the application **xclock** reads the defaults file **/usr/X11/lib/app-defaults/XClock**, which consists of exactly one line:

```
XClock.input: false
```

Note that because this resource is in a defaults file, the resource

```
*input: false
```

would behave exactly the same. If you wanted, for whatever reason, to permit a user to type input into **xclock**, edit this line to read:

```
XClock.input: true
```

As noted earlier, the resources in the X server's resource data base take precedence over the contents of a defaults file. For example, the X client **xgas** reads the default file **/usr/X11/lib/app-defaults/XGas**, which (among many others) contains the following resource:

```
*quit.label: Quit
```

This resource defines the text that appears on the "quit" button. If you decide that you want this button to be labelled **FOO**, you can insert the following resource into **$HOME/.Xdefaults:**

```
xgas*quit.label:FOO
```

Then, type the command:

```
xrdb -merge < $HOME/.Xdefaults
```

The next time you invoke **xgas**, the "quit" button will be labelled **FOO**, thus overriding the setting in **XGas**.

Modifying a Font Resource

Fonts are an important aspect of X. A font incorporates textual information; thus, a well-selected font can make your system much more useful.

For example, the defaults file **/usr/X11/lib/app-defaults/XTerm** sets a number of different fonts for using in different situations. The default font is called **fixed**. If you wish to change this to a larger font, try the following:

- **cd** to directory **/usr/X11/lib/fonts**. This directory holds the fonts available to the X system. These are kept two sub-directories: **misc** and **75dpi**. The former holds "miscellaneous" fonts, such as the **cursor** font; the latter holds fancier fonts built in a 75 dots-per-inch format. Most of the commonly used fonts are in **misc**.
- **cd** to **misc**. Read file **fonts.alias**. This gives the commonly used aliases for fonts used on the system. As you can see, the font named **fixed** is actually the font:

  ```
  -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
  ```

  When you look further in **fonts.alias**, you will see that this is the same as the font named **6x13**. The alias indicates that the font is 13 pixels high and 6 pixels wide. (For information on how to interpret the full, 14-field font name, see the Lexicon entry for **xfontsel**.) To select a larger font, pick one whose height is greater than 13 pixels or whose width is greater than six. For example, the font whose alias is **8x13** is the same height as the **fixed** font, but is two pixels wider.

- Close **fonts.alias**. Then edit your **.Xdefaults** file to insert the following entry:

  ```
  xvt*font:8x13
  ```

  This sets the font resource for **xvt** to the font with the alias **8x13**.

- Type:

  ```
  xrdb -merge < $HOME/.Xdefaults
  ```

  This merges your new resource setting into the X server's resource data base.

That's all there is to it. The next time you invoke **xvt**, it will use font **8x13**, which is slightly larger and more legible. Because the geometry of the **xvt** window is set to 80×25 (that, 80 columns by 25 rows), the window will be resized automatically to use the new font.

Note, by the way, that an **xvt** window using the **8x13** font will not completely fit onto a 640×480 screen — the last column slips off the right edge of the screen. If you find this to be a real problem, try using a narrower font.

Where To Go From Here

This concludes our introduction to X applications and how to customize them. We could only scratch the surface of resources, widgets, and the internals of X; however, we hope that you now know enough to make minor modifications to your system, and to begin to learn more about X.

For more information on a given application, see its entry in the Lexicon. The books referenced in the introduction will also give you more information.

The next chapter discusses how to recompile X applications under X Windows for COHERENT.

Recompiling X Applications

A wealth of X source code is available to the public. In all probability, somebody has already written the X program that you want. By checking publically accessible archives, you will find many interesting, useful, and amusing programs. The following gives hints on how to recompile X source code under COHERENT.

In many cases, importing an X application to COHERENT is simply matter of recompiling its source code. This section describes how to do so. It also discusses some problems that are commonly encountered during recompilation.

Imakefiles and Makefiles

The UNIX/COHERENT application **make** manages the building of programs. It reads the contents of a **Makefile**, which describes how to build the program, and names the source modules from which the program is created. If you are not familiar with **make**, see tutorial for it that appears in your COHERENT manual.

Although **make** is the programmer's best friend, adapting a **Makefile** from one operating system to another can be difficult. This is because different operating systems — even different flavors of UNIX or UNIX-like systems like COHERENT — can vary quite a bit in their capacities, in the functions available in their libraries, and in how they have structured their header files and libraries.

To help you avoid this problem, most X applications use the utility **imake** to build a **Makefile**. **imake** reads the contents of an **Imakefile** that is written by the programmer, invokes the C preprocessor **cpp** to combine the **Imakefile** with a set of configuration files designed for your operating system, and writes a **Makefile** from which you can build the application. In many instances, the **Imakefile** simply names the application you wish to build and the source files from which it is built. An example **Imakefile** is shown below.

The script **xmkmf** invokes **imake** with the arguments and configuration files that are appropriate for COHERENT. In most instances, all you have to do to build a **Makefile** is type **xmkmf**; and once the **Makefile** is built, all you have to do to build the application is type **make**.

Naturally, some problems may arise during this process. The commonest ones are discussed below.

Modifications to Makefiles

Some applications come with a **Makefile** instead of an **Imakefile**. If this is the case, you probably will need to modify this **Makefile** so that the program will compile correctly under COHERENT.

The following changes must be made to most **Makefile**s:

- X Windows for COHERENT keeps X header files in directory **/usr/X11/include/X11**. Most other releases of X keep their header files in directory **/usr/include/X11**; therefore, you must add an instruction to the **Makefile** to tell it to look in the correct directory.

  A **Makefile** must use the option -I to name this directory explicitly:

  ```
  -I/usr/X11/include
  ```

  This tells the C compiler to look in **/usr/X11/include** for any header file whose name is prefixed with X11, such as:

  ```
  #include <X11/Xos.h>
  ```

  Often, this is done as part of the macro **CCFLAGS**. Therefore, if you see an instruction of the form:

  ```
  -I/usr/include/X11
  ```

  change it to read:

  ```
  -I/usr/X11/include
  ```

- X Windows for COHERENT keeps its X libraries in directory **/usr/X11/lib**. A **Makefile** must name that directory explicitly. Many **Makefile**s use a macro named something like **LIBFLAGS** to set linking options and name libraries. Before the list of libraries, you must add the option:

```
-L/usr/X11/lib
```

- Add **-lXbsd** to the end of the list of libraries to be linked into the application. This tells the linker to link in library **libXbsd.a**, which holds the socket-emulation routines.
- If the option **-lsocket** appears on the list of libraries, remove it. This library holds the Berkeley socket functions; these are emulated in **libXbsd.a** and so are not needed.
- Add the option **-DCOHERENT** to the macro **CCFLAGS**. If you make any COHERENT-specific changes to the source code, you can bracket them with the preprocessor directives

```
#ifdef COHERENT
    ...
#endif
```

  and the instruction **-DCOHERENT** will ensure that they appear in the compiled program.

Once you have made these changes, you can begin compiling.

Problems Seen During Compilation

The following discusses problems that can come up during compilation.

- The COHERENT compiler is not ANSI compliant, although it recognizes some ANSI extensions to the C language. If the X application is written using ANSI-specific grammar (in particular, function prototypes), you must compile it with an ANSI-compliant compiler, e.g., GNU C. When doing so, modify the **Makefile** or **Imakefile** to define the macro **CC** to **gcc**.
- During the link phase, you may see the linker complain about the undefined symbol **select**. This indicates that one of the source modules has called the socket function **select()**. Under X Windows for COHERENT, this function is named **soselect()**.

  To fix this problem, go through the sources and conditionally replace every instance of

```
select( [arguments] );
```

  with:

```
#if defined(COHERENT)
        soselect ( [arguments] );
#else
        select ( [arguments] );
#endif
```

  Note that this is the proper way to add new COHERENT-specific code to X sources. If the application comes only with a **Makefile**, you must make sure the instruction **-DCOHERENT** appears as part of the macro **CFLAGS**; however, this is not necessary if you have built a **Makefile** from an **Imakefile**, as **xmkmf** ensures that this instruction is included automatically.
- The COHERENT C compiler does *not* include by default the code to print floating-point numbers, as this code increases the size of the linked executable noticeably. If your X application uses any of the options **%e**, **%f**, or **%g** with the function **printf()**, and if you are compiling with COHERENT's **cc** command, you *must* add the option **-f** to the macro **CFLAGS**. Note that you must do this in either a **Makefile** or an **Imakefile**, as **xmkmf** does not add this option by default.

  You do not need to do this if you are compiling with GNU C.
- Some applications redeclare the manifest constant **PI**; COHERENT declares this constant in header file **<math.h>**. The COHERENT C compiler will abort if manifest constants are redeclared (GNU C does not); so if this problem arises, conditionalize out the declaration of **PI** within the application, as shown above.
- You may find problems with the following routines: **bcopy()**, **bcmp()**, **bzero()**, **index()**, and **rindex()**. The problems usually consist of a clash between the manner in which these routines are declared or defined within the application, and they way they are declared or defined in the COHERENT header files. In most instances, you should conditionalize out the declaration within the application, and ensure that the proper COHERENT header file is included. For details, use the command **man** command to view the manual entries for the routine in question.

- Some versions of UNIX declare string-handling functions in the header file **<strings.h>**. COHERENT, however, keeps them in **<string.h>**. If the compiler complains that it cannot open **strings.h**, conditionally replace it in the sources with **<string.h>**, as described above.
- If you are compiling with GNU C, the linker may complain of a number of undefined symbols of the form **_dmul**. This is due to the fact that GNU C always creates code that performs hardware floating-point arithmetic and by default attempts to link in the COHERENT libraries that contain hardware floating-point routines, but the COHERENT X libraries were compiled to use software floating-point arithmetic. To get around this problem, you must modify the file **specs** for GNU C: change the line that reads

  ```
  %scrt1.o %scrtbegin.o -u _dtefg -L/lib/ndp -L/usr/lib
  ```

  to:

  ```
  %scrt1.o %scrtbegin.o -u _dtefg -L/lib -L/usr/lib
  ```

  Note that this is a work-around for GNU C, and may create problems of its own. For example, when you link software floating-point code with modules compiled to use hardware floating-point arithmetic, the function **atof()** will always return **NAN**.

  For a fuller description of how COHERENT manages floating-point arithmetic, see the Lexicon entry for **cc**.

If other problems arise that are not described here, please send a detailed description to MWC Technical Support, as described at the beginning of this manual.

Recompiling an Example Application

The following walks you through the recompilation of the X application **xwave**. This application, whose source code is included with X Windows for COHERENT, was written by Mike Friedman, Paul Riddle, and Jim McBeath. It draws a three-dimensional animation of a wave. By setting command-line parameters, you can dictate the size of the grid used and the type of wave to be plotted. The following steps describe how to recompile this program:

- **cd** to directory **/usr/X11/src**.
- De-archive the sources by typing the command:

  ```
  gtar -xvzf xwave.gtz
  ```

- **cd** to directory **xwave**.
- Type the command **xmkmf**. This builds a **Makefile** from the **Imakefile** included with this package.
- To recomile, type **make**.
- When compilation has finished, install the program as follows:

  ```
  mv xwave /usr/X11/bin
  ```

- To test the program, invoke X as described in an earlier section; then type **xwave**. The application opens a window and animates a sample wave.

That's all there is to it. A formatted manual page for **xwave** is included in directory **/usr/X11/src/xwave/man**.

The following gives the contents of **xwave**'s **Imakefile**:

```
LOCAL_LIBRARIES = $(XLIB)
OBJS = xwave.o force.o plot.o prop.o
SRCS = xwave.c force.c plot.c prop.c
SYS_LIBRARIES = -lm
ComplexProgramTarget(xwave)
```

When **xmkmf** and **imake** process this five-line **Imakefile**, it expands in a 370-line **Makefile**. The following describes each line in the **Imakefile**.

LOCAL_LIBRARIES names the libraries to be linked into the program. The macro **XLIB** means that the programmer wants to include the standard suite of X libraries for your system. The configuration files included with X Windows for COHERENT give the information that **imake** needs to expand **XLIB** into the correct set of libraries.

OBJS and **SRCS** name, respectively, the object modules from which the application is built, and the source files from which those objects are compiled.

SYS_LIBRARIES names the system libraries to be included, apart from the X libraries. The programmer has included the argument **-lm**, which indicates that the program needs routines within the mathematics library **libm.a**. For details on the contents of this library, see its entry in the COHERENT Lexicon.

Finally, the line

```
ComplexProgramTarget(xwave)
```

is a macro whose body is kept in file **/usr/X11/lib/config/Imake.rules**. This macro contains all of the instructions and commands that **make** needs to build a complex X program. The argument **xwave** names the application to be built.

For a discussion of the macros that can be included in an **Imakefile**, see the Lexicon entry for **imake** in this manual.

Building Your Own Makefile

If you do not want to use **xmkmf** and **imake**, you can build a **Makefile** by hand. You probably will never need to do this, but knowing how to do so may come in handy some time:

1. Build a skeletal **Makefile**. Place the following declarations at its beginning:

```
CFLAGS  = -I/usr/X11/include
LIBS    = -L/usr/X11/lib -lX11 -lXbsd
```

Follow this with a declaration of the object modules from which this application is built. As a rule of thumb, there is one object for each source module. For example, if the application consists of source modules **foo.c**, **bar.c**, and **baz.c**, write the following into your **Makefile**:

```
OBJS    = foo.o bar.o baz.o
```

Finally, add the target line for the executable you wish to build. If the application is to be called **xapp**, add the following to your skeletal **Makefile**:

```
xapp: $(OBJS)
       $(CC) $(CFLAGS) $(OBJS) $(LIBS)
```

2. Use the command

```
cc -c -I/usr/X11/include/X11 source.c
```

for each *source* module. Some modules will compile correctly on the first try; others will require several attempts. Those that require several attempts may require that you use the **cc** option **-D** to set one or more switches within the source module. Use the command **grep** to find every instance with the source module of the C preprocessor directive **#if**. These will indicate the options that are available within the source code, and suggest which switches you should set.

3. Once you have succeeded in compiling all of the source files to objects, type **make**. Because all of the objects already exist, **make** will attempt to link an executable.

At this stage, you probably will see errors about undefined symbols. Note which symbols are undefined; then use the COHERENT utility **nm** to list the symbols in all of the X libraries and find which libraries contains the symbols you need. Include them in your **Makefile**.

Note that an X application always need the libraries **libX11.a** and **libXbsd.a**; the declarations given above invoke those libraries automatically. The difficulty is in arranging the other libraries in the proper order in the **Makefile**. This can be done only by trial and error: sometimes many trials and many errors. When you have the correct order, copy the information into your **Makefile**.

Unresolved identifiers within the program may also be manifest constants that the programs expects to be set with the **-D** option to the **cc** command.

When you have finished linking the program, you should have a working executable. Thereafter, you can use the **Makefile** you just created to rebuild the application, should you decide to modify the source code.

Most applications come with a manual page that describe the program and how to run it. You may, for example, need to install a font or a resource file in the appropriate directory. You will, of course, need to test the program to make sure that it runs correctly. We have found that, in most instances, if a program can be compiled and linked under COHERENT, it will also run correctly.

Where To Get X Sources

Numerous X programs have already been ported to COHERENT. Archives of sources are available for free on the Mark Williams BBS. For directions on how to contact the MWC BBS, see the directions in the tutorial for UUCP that appears in the COHERENT manual.

If you have access to the Internet, you can retrieve source files from the site **raven.alaska.edu**. Use **ftp** to access that site; then log in as "anonymous". New sources are added continually.

The master site for X software is **ftp.x.org**. If you have access to the Internet, you can log into that system via anonymous **ftp**. This site contains many megabytes of sources, so you would be well advised to think about what you would like to retrieve before you enter this site.

Finally, Mark Williams Company sells packages of X applications that have been ported to COHERENT. Called **Xware**, these packages bring together interesting, useful, and amusing programs for your COHERENT X system. Each has been ported to COHERENT; most include full source code and a formatted manual page that can be viewed with the COHERENT **man** command. For details on **Xware**, see the release notes that come with this manual.

The Lexicon

The rest of this manual consists of a Lexicon. This Lexicon contains one article for each of the utilities and clients included with X Windows for COHERENT. The articles appear in alphabetical order.

appres — X Utility

List an application's resource data base
appres [[*class* **[***instance***]] [-1]**

appres prints the resources that are seen by an application (or subhierarchy of an application) and have *class* and *instance*. You can use it to determine which resources a particular program will load. For example, the command

```
appres XTerm
```

lists the resources that the command **xterm** loads. If you specify no application class, **appres** uses the class **AppResTest**.

To match a particular instance name, specify that name explicitly after the class name; for example:

```
appres XTerm myxterm
```

To list resources that match a subhierarchy of an application, name the hierarchical classes and instance names. The number of class and instance components must be equal. To list just the resources that match a specific level in the hierarchy, use the option **-1**. For example, the command

```
appres XTerm.VT100 xterm.vt100 -1
```

lists the resources that match the **xterm** widget **vt100**.

See Also

xprop, xrdb, X utilities, xwininfo

Notes

appres was written by Jim Fulton of the MIT X Consortium.

atobm — X Utility

Convert ASCII to an X bit-mapped image
atobm [-chars *cc***] [-name** *variable***] [-xhot** *number***] [-yhot** *number***] [***file***]**

atobm converts ASCII text into an X bit-mapped image. *file* must hold a bit-mapped image that had been created with the editor **bitmap** and had been converted into ASCII by the utility **bmtoa**. For details on what constitutes a bit-mapped image, see the Lexicon entry for **bitmap**. For an example of using a bit-mapped image with the X server, see the Lexicon entry for **xsetroot**.

file gives the file in which the text resides. If no *file* appears on the command line, **atobm** reads the standard input.

atobm recognizes the following command-line options:

-chars *cc*
: Specify the pair of characters to use when converting strings into bits. The first character of the pair represents the zero bit, and the second represents the one bit. The default is to use a hyphen '-' to represent zero and a pound sign '#' to represent one.

-name *variable*
: Set to *variable* the name used when writing out the bit-map file. The default is to use the base name of the command-line argument *file* (or leave it blank if **atobm** is reading the standard input).

-xhot *number*
: Give the X coordinate of the image's "hot spot." The hot spot is the pixel that, on an image used as a mouse cursor, identifies exactly where the image is pointing. Only positive values are allowed. By default, a bit-mapped image includes no hot-spot.

-yhot *number*
: Give the Y coordinate of the image's "hot spot." Only positive values are allowed.

Example

Consider following the bit-mapped image:

```
---------------
-------#-------
-------#-------
---#---#---#----
----#--#--#-----
-----#-#-#------
------#-#-------
-#####---#####--
------#-#-------
-----#-#-#------
----#--#--#-----
---#---#---#----
-------#-------
-------#-------
---------------
---------------
```

You may have created this image with the bit-map editor **bitmap**, or you may simply have created it with a text editor, such as MicroEMACS. If you have saved it in a file called **star**, you can filter it through **atobm** via the following command:

```
atobm star
```

This produces the following:

```
#define star_width 16
#define star_height 16
static char star_bits[] = {
   0x00, 0x00, 0x80, 0x00, 0x80, 0x00, 0x88, 0x08, 0x90, 0x04, 0xa0, 0x02,
   0x40, 0x01, 0x3e, 0x3e, 0x40, 0x01, 0xa0, 0x02, 0x90, 0x04, 0x88, 0x08,
   0x80, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00};
```

Your bit-mapped image is now ready for loading into an X application.

See Also

bitmap, bmtoa, X utilities

Notes

atobm was written by Davor Matic of the X Consortium.

bdftopcf — X Utility

Generate a PCF font from a BDF file
bdftopcf [-i] [-L] [-l] [-M] [-m] [-o *pcf_file***] [-p***number***] [-t] [-u***number***] [***bdf_file***]**

bdftopcf converts a font from bitmap distribution format (BDF) to the portable compiled format PCF normally used by X Windows for COHERENT. *bdf_file* names the file that holds the BDF font; if none is named, **bdftopcf** reads the standard input.

bdftopcf recognizes the following command-line options:

-i Do not compute "ink metrics" for terminal-emulator fonts.

-L Force the byte order to least-significant-byte first.

-l Force the bit order to least-significant-bit first.

-M Force the byte order to most-significant-byte first.

-m Force the bit order to most-significant-bit first.

-o *pcf_file*
Write the output into *pcf_file*. If this command-line option is not used, **bdftopcf** writes the PCF file to the standard output.

-p*number*
Force the glyph padding to *number*, which can be one, two, four, or eight.

-t Expand the glyphs in the terminal-emulator fonts to fill the bounding box.

-u*number*
Force the scanline-unit padding to *number*, which must be one of one, two, or four.

See Also

mkfontdir, X utilities

bitmap — X Utility

Bit map editor
bitmap [*-options***] [***filename***] [***basename***]**

bitmap is a rudimentary tool for creating or editing rectangular images made up of 1's and 0's. X uses bit-maps to define clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

Using bitmap

bitmap displays a grid, each square of which represents one pixel in the picture being edited. To see the bit-mapped image in its actual size, as it appears both normally and inverted, press **<ctrl-I>**. You can drag the popped-up image out of the way to continue editing. To remove the bit-mapped image, either move the mouse cursor into the pop-up window and press the left mouse button, or press **<ctrl-I>** again.

If you wish to use the bit-mapped image to define a mouse cursor, you can designate one of the squares in the image as the "hot spot" — that is, the point in the cursor that determines exactly where the cursor is pointing. For cursors with sharp points (e.g., an arrow or finger), the hot spot is usually at the tip; for symmetric cursors (such as a cross or bull's-eye), it usually is at the center.

bitmap stores a bit-map as a fragment of C code that is suitable for loading in applications. The C code gives an array of bits as well as symbolic constants that give the width, height, and hot spot (if specified) that can be used to create cursors, icons, and tiles. For examples of the output of **bitmap**, see the Lexicon entry for **bmtoa**.

Editing

To edit a bit-mapped image, simply click on one of the buttons with drawing commands and move the mouse cursor into the bit-map grid window. When you press one of the buttons on your mouse, **bitmap** performs the appropriate action. You can set, clear, or invert a grid square. Setting a square corresponds to setting a bit in the bit-mapped image to one; clearing a square corresponds to setting a bit to zero. Inverting a grid square corresponds to changing a bit from zero to one or one to zero. The following gives the default behavior of the mouse buttons:

MouseButton1 (left mouse button):	Set
MouseButton2 (center mouse button):	Invert
MouseButton3 (right mouse button):	Clear

This default behavior can be changed by setting the mouse buttons' function resources. The following example inverts the default behavior of the center and right mouse buttons:

```
bitmap*button1Function: Set
bitmap*button2Function: Clear
bitmap*button3Function: Invert
```

The button function applies to all drawing commands, including copying, moving, and pasting, flood filling, and setting the hot spot.

Drawing Commands

The following lists the buttons on the left side of **bitmap**'s window. Each accessing a drawing command. You can abort some commands by pressing 'A' inside the bit-map window:

`(Circle)` Change the grid squares in a circle between two squares. Once you press a mouse button in the grid window, **bitmap** highlights the circle from the square where the mouse button was initially pressed to the square where the mouse cursor is located. Releasing the mouse button causes the change to take effect, and the highlighted circle disappears.

`(Clear)` Clear all bits in the bit-mapped image. The grid squares are set to the background color. Pressing 'C' inside the window has the same effect.

(Clear_Hot_Spot)
Remove any designated hot spot from the bit-mapped image.

(Copy) Copy an area of the grid from one location to another. If no marked grid area is displayed, This command behaves like **Mark**, described below.

Once a marked grid area is displayed in the highlighting color, this command has two alternative behaviors. If you click a mouse button inside the marked area, you can drag the rectangle that represents the marked area to the desired location. After you release the mouse button, the area is copied. If you click outside the marked area, **Copy** assumes that you wish to mark a different region of the bit-map image; thus, it again behaves like **Mark**.

(Curve) Change the grid squares underneath the mouse cursor if a mouse button is being pressed down. If you drag the mouse continuously, **bitmap** ensures that the line is continuous. If your system is slow or **bitmap** receives very few mouse motion events, it might behave quite strangely.

(Down) Moves the bit-map image one pixel down. If a marked area of the grid is highlighted, it operates only within the marked area. Pressing (°) inside the window has the same effect.

(Filled_Circle)
This command is identical to **Circle**, except that the circle is filled rather than outlined.

(Filled_Rectangle)
Identical to **Rectangle**, except at the end the rectangle is filled rather than outlined.

(Flip_Horizontally)
Flip the bit-map image along the horizontal axis. If a marked area of the grid is highlighted, this command operates only within the marked area. Pressing 'F' inside the window has the same effect.

(Flip_Vertically)
Flip the bit-mapped image along the vertical axis. If a marked area of the grid is highlighted, this command operates only within the marked area. Pressing 'V' inside the window has the same effect.

(Flood_Fill)
Flood-fill the connected area beneath the mouse cursor when you click on the desired square. Diagonally adjacent squares are not considered to be connected.

(Fold) Fold the bit-mapped image so that the opposite corners become adjacent. This is useful when creating bit-mapped images for tiling. Pressing 'F' inside the window has the same effect.

(Invert) Invert all bits in the bit-mapped image. The grid squares are inverted appropriately. Pressing 'I' inside the window has the same effect.

(Left) Moves the bit-mapped image one pixel to the left. If a marked area of the grid is highlighted, it operates only within the marked area. Pressing (æ) inside the window has the same effect.

(Line) Change the grid squares in a line between two squares. When you press a mouse button in the grid window, **bitmap** highlights the line from the square where the mouse button was initially pressed to the square where the mouse cursor is located. Releasing the mouse button causes the change to take effect, and the highlighted line disappears.

(Mark) Mark an area of the grid by dragging out a rectangular shape in the highlighting color. Once the area is marked, it can be operated on by commands **Up**, **Down**, **Left**, **Right**, **Rotate**, **Flip**, or **Cut**. Only one marked area can be present at any time. If you attempt to mark another area, the old mark will vanish. The same effect can be achieved by simultaneously pressing **<shift>** and the left mouse button, and dragging out a rectangle in the grid window. Simultaneously pressing **<shift>** and the middle mouse button marks the entire grid area.

(Move) Move an area of the grid from one location to another. Its behavior resembles the behavior of the command **Copy**, except that the marked area is moved instead of copied.

(Point) Change the grid squares underneath the mouse cursor if a mouse button is being pressed down. If you drag the mouse continuously, the line may not be continuous, depending on the speed of your system and frequency of mouse-motion events.

(Rectangle) Change the grid squares in a rectangle between two squares. When you press a mouse button in the grid window, **bitmap** highlights the rectangle from the square where the mouse button was initially pressed to the square where the mouse cursor is located. Releasing the mouse button causes the

change to take effect, and the highlighted rectangle disappears.

(Right) Moves the bit-mapped image one pixel to the right. If a marked area of the grid is highlighted, it will operate only within the marked area. Pressing (Æ) inside the window has the same effect.

(Rotate_Left)
Rotate the bit-map image 90˚ to the left (counter-clockwise). If a marked area of the grid is highlighted, it will operate only within the marked area. Pressing 'L' inside the window has the same effect.

(Rotate_Right)
Rotate the bit-mapped image 90˚ to the right (clockwise). If a marked area of the grid is highlighted, it operates only within the marked area. Pressing 'R' inside the window has the same effect.

(Set) Set all bits in the bit-mapped image. The grid squares are set to the foreground color. Pressing 'S' inside the window has the same effect.

(Set_Hot_Spot)
Designate one square in the grid as the hot spot. Pressing a mouse button in the desired square displays a diamond shape.

(Undo) Undo the last executed command. It has depth one — that is, pressing **Undo** after **Undo** will undo itself.

(Unmark) Erase the marked area. The same effect can be achieved by simultaneously pressing **<shift>** and the right mouse button.

(Up) Move up the bit-mapped image by one pixel. If a marked area of the grid is highlighted, this command operates only within the marked area. Pressing (ª) inside the window has the same effect.

File Menu

To access the File Menu's commands, click the (File) button and select the appropriate menu entry, or press **<ctrl>** key with another key. These commands deal with files, and with global parameters of the bit map (e.g., size, basename, and file name):

(Basename) Change the base name, if you want one other than the specified file name.

(Filename) Change the file name without changing the base name or saving the file. If you specify '-' for a file name, **bitmap** writes its output to the standard output.

(Insert) Insert a bit-map image into the image being currently edited. After **bitmap** has prompted you for the name of the file that holds the bit-mapped image to insert, click inside the grid window and drag the outlined rectangle to the point where you wish to insert the newly loaded image.

(Load) Load a new bit-mapped image into the editor. If you have not yet saved the current image, **bitmap** asked you whether to save the changes. The editor can edit only one file at a time. If you need interactive editing, run a number of editors and use the cut-and-paste mechanism, as described below.

(New) Clear the editing area and prompt for the name of the new file to edit. It will not load in the new file.

(Quit) Terminate **bitmap**. If the file was not saved, **bitmap** prompts you to ask whether to save the image. This command is preferred over killing the process.

(Rescale) Rescale the editing area to the new width and height. Enter the size in the format *width*×*height*. This commands does not do antialiasing, and information is lost if you rescale to the smaller sizes.

(Resize) Resize the editing area to the new number of pixels. Enter the size in the format *width*×*height*. The information in the image being edited will not be lost unless the new size is smaller that the current image size. NB, the editor was not designed to edit huge files.

(Save) Save the bit-map image. It does not prompt for the file name unless it is said to be **<none>**. If you leave the file name undesignated or '-', **bitmap** writes the image to the standard output.

(Save_As) Save the bit-mapped image after prompting for a new file name. It should be used if you want to change the file name.

Edit Menu

To access the Edit Menu's commands, press the (Edit) button and select the appropriate menu entry, or press **<esc>** with another key. These commands deal with editing facilities, such as grid, axes, zooming, and cut and paste:

(Axes) Highlight the main axes of the image being edited. These lines are not part of the image, but are provided to help you keep your images symmetrical.

(Copy) Copy the contents of the highlighted image area into the internal cut-and-paste buffer.

(Cut) Cut the highlighted image area into the internal cut-and-paste buffer.

(Dashed) Control the stipple for drawing the grid lines. The stipple specified by the resource **dashes** can be turned on or off by activating this command.

(Grid) Control the grid in the editing area. If the grid spacing is less than that specified by the resource **gridTolerance** (default, eight), **bitmap** automatically turn the grid off.

(Image) Open a separate window and display therein the image being edited and its inverse in their actual size. You can move the new window. To erase the new window, move the mouse cursor into it and press the left mouse button.

(Paste) Copy into the current image either the highlighted area from another image or the contents of **bitmap**'s internal cut-and-paste buffer. To place the copied image, click in the editing window, drag the outlined image to the position where you want to place it, then release the button.

(Proportional) Toggle proportional mode. If proportional mode is on, **bitmap** forces all squares to have the same width and height, regardless of the proportions of the bit-map window.

(Stippled) Toggle stippling of the highlighted areas of the bit-mapped image. The stipple is specified by resource **stipple**.

(Zoom) Toggle zoom mode. If an area of the image is highlighted, **bitmap** automatically zooms into it. Otherwise, **bitmap** lets you highlight an area to be edited in the zoom mode; **bitmap** then automatically switches into it. You can use all the editing commands and other utilities in the zoom mode. When you zoom out, the command **undo** undoes the entire zoom session.

Cut and Paste

bitmap supports two cut-and-paste mechanisms: the internal cut-and-paste, and the global X selection cut-and-paste.

bitmap uses the internal cut-and-paste feature when it executes the drawing commands **copy** and **move**, and when it executes the commands **cut** and **copy** on its **Edit** menu. It uses the global X selection cut-and-paste whenever any area of a bit-mapped image is highlighted.

To copy part of an image from another bit-map editor, simply use the command **Mark** to highlight the desired area, or pressing the **<shift>** key and drag the area with the left mouse button. When X highlights the selected area, any other application (such as **xterm**) that use the primary selection will discard its selection values and unhighlight the appropriate information. To drop the cut portion of an image into the image you are now editing, use the **Paste** command from the **Edit** menu, or press the mouse's control button.

If you attempt to do this without a visible highlighted image area, **bitmap** reads its internal cut-and-paste buffer and pastes whatever is stored there at the moment.

Widgets

Below is the widget structure of the **bitmap** application. Indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name. All widgets except the **bitmap** widget are from the standard Athena widget set.

```
Bitmap bitmap
      TransientShell image
            Box box
                  Label normalImage
                  Label invertedImage

      TransientShell input
            Dialog dialog
                  Command okay
                  Command cancel
```

```
TransientShell error
      Dialog dialog
            Command abort
            Command retry

TransientShell qsave
      Dialog dialog
            Command yes
            Command no
            Command cancel

Paned parent
      Form formy
            MenuButton fileButton
            SimpleMenu fileMenu
                  SmeBSB  new
                  SmeBSB  load
                  SmeBSB  insert
                  SmeBSB  save
                  SmeBSB  saveAs
                  SmeBSB  resize
                  SmeBSB  rescale
                  SmeBSB  filename
                  SmeBSB  basename
                  SmeLine line
                  SmeBSB  quit

            MenuButton editButton
            SimpleMenu editMenu
                  SmeBSB  image
                  SmeBSB  grid
                  SmeBSB  dashed
                  SmeBSB  axes
                  SmeBSB  stippled
                  SmeBSB  proportional
                  SmeBSB  zoom
                  SmeLine line
                  SmeBSB  cut
                  SmeBSB  copy
                  SmeBSB  paste
            Label status
```

```
Pane pane
      Bitmap bitmap
      Form form
            Command clear
            Command set
            Command invert
            Toggle  mark
            Command unmark
            Toggle  copy
            Toggle  move
            Command flipHoriz
            Command up
            Command flipVert
            Command left
            Command fold
            Command right
            Command rotateLeft
            Command down
            Command rotateRight
            Toggle  point
            Toggle  curve
            Toggle  line
            Toggle  rectangle
            Toggle  filledRectangle
            Toggle  circle
            Toggle  filledCircle
            Toggle  floodFill
            Toggle  setHotSpot
            Command clearHotSpot
            Command undo
```

Colors

If you want **bitmap** to be viewable in color, include the following in the **#ifdef COLOR** section of the file you read with **xrdb**:

```
*customization: -color
```

This tells **bitmap** to pick up the colors in the color-customization file **/usr/X11/lib/app-defaults/Bitmap-color**.

Command-line Options

bitmap recognizes the following command-line arguments:

-axes Turn the major axes off.

+axes Turn the major axes on.

basename Specify the base name to use in the output file of C code. If it differs from the base name in the working file, **bitmap** changes it when saving the file.

-dashed [*filename***]**
Turn off dashing for the frame and grid lines. *filename* names the bit-map to use as a stipple.

+dashed Turn on dashing for the frame and grid lines.

filename Name the bit-map to be initially loaded into the program. If *filename* does not exist, **bitmap** assumes it is a new file.

-fr *color* Use *color* for the frame and grid lines. For a list of colors that X recognizes, see file **/usr/X11/lib/rgb.txt**.

-grid Turn the grid off.

+grid Turn the grid on.

-gt *dimension*
Set grid tolerance. If the square dimensions fall below *dimension*, **bitmap** automatically turns the grid off.

LEXICON

-hl *color* Use *color* for highlighting.

-proportional
Turn off proportional mode. If proportional mode is on, square width equals square height. If proportional mode is off and the square's dimensions differ, **bitmap** uses the smaller dimension.

+proportional
Turn on proportional mode.

-sh *pixels* Set the height of squares, in *pixels*.

-size *width×height*
Set the size of the grid in squares.

-stipple *filename*
Set the bit-map to be used as a stipple for highlighting.

-stippled Turn off stippling of highlighted squares.

+stippled Turn on stippling of highlighted squares.

-sw *pixels* Set the width of squares, in *pixels*.

Bitmap Widget

The widget **bitmap** is a stand-alone widget for editing raster images. It is not designed to edit large images, although you can use it for that purpose as well. You can incorporate it with other applications and use it as a standard editing tool. The following are the resources provided by the widget **bitmap**:

```
Bitmap Widget

Header file Bitmap.h
Class                   bitmapWidgetClass
Class Name              Bitmap
Superclass              Bitmap
```

This widget uses all of the simple-widget resources, plus the following:

Name	*Class*	*Type*	*Default Value*
foreground	Foreground	Pixel	XtDefaultForeground
highlight	Highlight	Pixel	XtDefaultForeground
framing	Framing	Pixel	XtDefaultForeground
gridTolerance	GridTolerance	Dimension	8
size	Size	String	32x32
dashed	Dashed	Boolean	True
grid	Grid	Boolean	True
stippled	Stippled	Boolean	True
proportional	Proportional	Boolean	True
axes	Axes	Boolean	False
squareWidth	SquareWidth	Dimension	16
squareHeight	SquareHeight	Dimension	16
margin	Margin	Dimension	16
xHot	XHot	Position	NotSet (-1)
yHot	YHot	Position	NotSet (-1)
button1Function	Button1Function	DrawingFunction	Set
button2Function	Button2Function	DrawingFunction	Invert
button3Function	Button3Function	DrawingFunction	Clear
button4Function	Button4Function	DrawingFunction	Invert
button5Function	Button5Function	DrawingFunction	Invert
filename	Filename	String	None ("")
basename	Basename	String	None ("")

See Also

atobm, bmtoa, editres, X utilities

Notes

The default screen for **bitmap** is extremely large. With its default resource file, it may not entirely fit onto a 640×480 screen.

bitmap was written by Davor Matic of the X Consortium.

bmtoa — X Utility

Convert an X bit-mapped image to ASCII
bmtoa [-chars *cc***] [***file***]**

bmtoa converts into ASCII an X bit-mapped image that had been created with the editor **bitmap**.

file names the file that holds the image. If no *file* is named on the command line, **bmtoa** reads the standard input.

The command-line option **-chars** specifies the pair of characters to use in the string version of the bit-mapped image. The first character of the pair *cc* represents zero bits, and the second represents one bits. The default is to use a hyphen '-' to represent zero and a pound sign '#' to represent one.

Example

Consider the file **/usr/X11/include/X11/bitmaps/star**, which reads as follows:

```
#define star_width 16
#define star_height 16
#define star_x_hot 7
#define star_y_hot 7
static char star_bits[] = {
   0x00, 0x00, 0x80, 0x00, 0x80, 0x00, 0x88, 0x08, 0x90, 0x04, 0xa0, 0x02,
   0x40, 0x01, 0x3e, 0x3e, 0x40, 0x01, 0xa0, 0x02, 0x90, 0x04, 0x88, 0x08,
   0x80, 0x00, 0x80, 0x00, 0x00, 0x00, 0x00, 0x00};
```

When you execute the command

```
bmtoa /usr/X11/include/X11/bitmaps/star
```

you see:

```
----------------
-------#--------
-------#--------
---#---#---#----
----#--#--#-----
-----#-#-#------
------#-#-------
-#####---#####--
------#-#-------
-----#-#-#------
----#--#--#-----
---#---#---#----
-------#--------
-------#--------
----------------
----------------
```

See Also

atobm bitmap, X utilities

Notes

bmtoa was written by Davor Matic of the X Consortium.

editres — X Utility

Resource editor for X Toolkit applications
editres [-*toolkitoption* **...]**

The X utility **editres** lets you view the full widget hierarchy of any X Toolkit client that speaks the **Editres** protocol. In addition, **editres** helps you construct a resource specification, then lets you apply the resource to the application and view the results. Once you are happy with a resource specification, **editres** appends the resource string to your X resources file.

Command-line Options

editres recognizes the following command-line options:

LEXICON

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host*[**:***server*][**.***screen*]
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

editres Window

editres displays a window that consists of the following four areas:

Menu Bar A set of pop-up menus that allow you full access to editres's features.

Panner This lets the user "pan" through an image. This is a more intuitive way to scroll the application tree's display.

Message Area
Display information about the action that **editres** expects of the user.

Application Widget Tree
Display the selected client's widget tree.

To begin an **editres** session, first select the menu item `(Get_Widget_Tree)` from the command menu. This changes the mouse cursor to a cross hair. Click on the window of the application you wish to examine. If this application understands the **Editres** protocol, then **editres** displays the client's widget tree in its tree window. If the application does not understand the **Editres** protocol, **editres** informs you of this fact in the message area.

editres Menu Commands

Once you have read a widget tree, you can manipulate it by selecting any of the following options from the **editres** menu:

`(Dump_Widget_Tree_to_a_File)`
This option dumps into a file a text description of the application's widget tree. The output of this option is especially useful when you wish to include an application's widget tree in a manual page. When you select this option, **editres** actives a pop-up dialogue. Type into the dialogue the name of the file into which you wish to dump the description; then either click the button labelled `(okay)` or press the `(¢)` key. **editres** dumps the widget tree to this file. To cancel the file dialogue, click the buttom labelled `(cancel)`.

`(Quit)` Exit **editres**.

`(Refresh_Widget_Tree)`
editres only knows about the widgets that an application was using at the moment you read its widget tree. Many applications create and destroy widgets "on the fly." This menu item tells **editres** to re-read an application's widget tree, thus updating its information to the current state of the application.

`(Send_Widget_Tree)`
Read the widget tree of any client that speaks the **Editres** protocol by clicking on any of its windows.

`(Set_Resource)`
This command displays a dialogue box in which you can set a resource for all selected widgets. Type the name of the resource and the value to which you are setting it. To jump between the resource-name field and the resource-value field, press the `(Tab)` key.

(Show_Resource_Box)
Display a resource box for the current client. This box (described in detail below) lets you see exactly which resources can be set for the widget that is currently selected in the widget tree's display. Only one widget can be selected at a time; if more or fewer are selected, **editres** refuses to display the resource box and displays an error message in the **Message Area**.

Tree Commands

The **Tree** menu contains the following commands with which you can manipulate the widget tree:

(Flash_Active_Widgets)
This command is the inverse of the command **Select Widget in Client**: for each widget that is currently selected in the widget tree, it flashes the corresponding widget in the application. It flashes each widget *numFlashes* times (default, three) in *flashColor*.

(Invert_All)
Invert every widget in the widget tree.

(Select_All)
Select all widgets in the widget tree.

(Select_Ancestors)
Select all parents of each of the currently selected widgets. This is a recursive search.

(Select_Children)
Select the immediate children of each of the currently selected widgets.

(Select_Descendants)
Select all children of each of the currently selected widgets, yea, even unto the last generation. This is a recursive search.

(Select_Parents)
Select the immediate parent of each of the currently selected widgets.

(Select_Widget_in_Client)
When you select this command, the mouse cursor turns into a cross hair. When you click a widget in the application, **editres** highlights the corresponding entry in the widget tree's display. Because some widgets are obscured by their children, it is not possible to display to every widget this way.

(Show_Class_Names)
Change the label of every widget in the tree to show the widget's class name instead of the widget's name.

(Show_Widget_IDs)
Change the label of every widget in the tree to show the widget's ID, in hexadecimal, instead of the widget's name.

(Show_Widget_Names)
Change the label of every widget in the tree to show the widget's name. This is the default.

(Show_Widget_Windows)
Change the label of every widget in the tree to show the widget's window, in hexadecimal, instead of the widget's name.

(Unselect_All)
Unselect all widgets in the widget tree.

Most of the above commands have a keyboard equivalent, as follows:

Key	*Command*	*Translation Entry*
space	Unselect	Select(nothing)
C	Show Class Names	Relabel(class)
I	Show Widget IDs	Relabel(id)
N	Show Widget Names	Relabel(name)
T	Toggle Widget/Class Name	Relabel(toggle)
W	Show Widget Windows	Relabel(window)
a	Select Ancestors	Select(ancestors)
c	Select Children	Select(children)
d	Select Descendants	Select(descendants)
i	Invert	Select(invert)
p	Select Parent	Select(parent)
s	Select	Select(all)
w	Select	Select(widget)

To add a widget to the set of selected widgets, click the left mouse button. To select a widget and un-select all other widgets, click the middle mouse button on that widget. Clicking the right mouse button on a widget toggles a widget's label between the widget's instance name and the widget's class name.

Using the Resource Box

The resource box contains five areas, as follows:

Resource Line
This area, which is at the top of the resource box, shows the name of the current resource exactly as it would appear if you were to save it into a file or apply it.

Widget Names and Classes
This area allows you to select the widgets to which this resource will apply. The area contains four lines, as follows:

1. The name of the selected widget and all its ancestors, and the more-restrictive period '.' separator.
2. The class names of each widget, and well as the less-restrictive asterisk '*' separator.
3. A set of special buttons labelled `(Any_Widget)`, which generalizes this level to match any widget.
4. A set of special buttons labelled `(Any_Widget_Chain)`, which turns the single level into something that matches zero or more levels.

The initial state of this area is the most restrictive, using the resource names and the period separator. By selecting the other buttons in this area, you can ease the restrictions to allow more and more widgets to match the specification. The extreme case is to select all the `(Any_Widget_Chain)` buttons, which matches every widget in the application. As you select different buttons, the tree's display updates to show you exactly which widgets are affected by the resource specification.

Normal and Constraint Resources
The next area lets you select the name of the normal or constraint resources you wish to set. Some widgets may not have constraint resources, so that area will not appear.

Resource Value
This next area allows you to enter the resource value. Enter this value exactly as you would type a line into your resource file; thus, it should contain no unescaped new-lines. There are a few special character sequences for this file:

\n A newline.
*****OOO*
A number, where *OOO* represents three octal digits. This is replaced by one byte that contains this sequence interpreted as an octal number. For example, a NUL can be represented by the sequence **\000**.
\<new-line>
A blank line.
\\ A literal backslash.

Command Area
This area contains the following command buttons:

(Apply) Perform a **XtSetValues()** call on all widgets that match the *resource line* described above. The value specified is applied directly to all matching widgets. This behavior is an attempt to give a dynamic feel to the resource editor. Because this feature lets you put an application into a state that it is not willing to handle, a hook has been provided to allow specific clients block these **XtSetValues()** requests (see the discussion of blocking editres requests, below).

Unfortunately, the X Toolkit and Resource Manager impose constraints on widgets; thus,trying to coerce an inherently static system into dynamic behavior can produce strange results. There is no guarantee that when you save a value and restart an application, that application behaves the same as it appeared under **editres**. This feature is provided to give you a rough feel for what your changes will do; the results should be considered suspect at best.

(Popdown_Resource_Box)
Remove the resource box from the display.

(Save) Append the *resource line* described above onto the end of the save file. If no save file has been set, **editres** displays the (Set_Save_File) dialogue box.

(Save_and_Apply)
Combine into one button the **Save** and **Apply** actions described above.

(Set_Save_File)
Change the file into which the resources are saved. When you press this button, **editres** displays a dialogue box that requests a file name. Once the file name has been entered, either press (¢) or click on the button labelled (okay).To close the dialogue box without changing the file, click the button labelled (cancel)

Blocking editres Requests

The **editres** protocol has been built into the Athena Widget set. This allows all application that are linked against **Xaw** to be able to speak to the resource editor. Although this provides great flexibility, and is a useful tool, it can quite easily be abused. It is therefore possible for any **Xaw** client to specify a value for the resource **editresBlock**, described below, to keep **editres** from divulging information about its internals, or to disable the **SetValues** part of the protocol.

editresBlock (Class **EditresBlock)**
Specify the type of blocking this client wishes to impose on the **editres** protocol. The accepted values are as follows:

all	Block all requests.
setValues	Block all **setvalues** requests. This is the only **editres** request that actually modifies the application; in effect, it states that the application is read-only.
none	Allow all **editres** requests.

Remember that these resources are set on any **Xaw** client, *not* **editres**. They allow individual clients to stop some or all of the requests that **editres** makes from ever succeeding.

Note that **editres** is also an **Xaw** client, so it can view and modify itself; these commands can be blocked by setting the **editresBlock** resource on **editres** itself.

Resources

editres uses the following application-specific resources:

numFlashes (Class **NumFlashes)**
Set the number of times **editres** flashes the widgets in the client application when you invoke the command **Show Active Widgets**.

flashTime (Class **FlashTime)**
Amount of time between the flashes described above.

flashColor (Class **flashColor)**
Set the color used when **editres** flashes clients. A bright color should be used, such as red or yellow — one that immediately draws your attention to the area being flashed.

saveResourcesFile (Class **SaveResourcesFile)**
The file into which the resource line is append to when the you select the **Save** button Resource box.

Widgets

The following gives the widgets that **editres** uses:

```
Editres  editres
      Paned  paned
            Box  box
                  MenuButton  commands

                        SimpleMenu  menu
                              SmeBSB  sendTree
                              SmeBSB  refreshTree
                              SmeBSB  dumpTreeToFile
                              SmeLine  line
                              SmeBSB  getResourceList
                              SmeLine  line
                              SmeBSB  quit

                  MenuButton  treeCommands
                        SimpleMenu  menu
                              SmeBSB  showClientWidget
                              SmeBSB  selectAll
                              SmeBSB  unselectAll
                              SmeBSB  invertAll
                              SmeLine  line
                              SmeBSB  selectChildren
                              SmeBSB  selectParent
                              SmeBSB  selectDescendants
                              SmeBSB  selectAncestors
                              SmeLine  line
                              SmeBSB  showWidgetNames
                              SmeBSB  showClassNames
                              SmeBSB  showWidgetIDs
                              SmeBSB  showWidgetWindows
                              SmeLine  line
                              SmeBSB  flashActiveWidgets

            Paned  hPane
                  Panner  panner
                  Label  userMessage
                  Grip  grip

            Porthole  porthole
                  Tree  tree
                        Toggle  <name of widget in client>
                              .
                              .
                              .
                              TransientShell  resourceBox
                                    Paned  pane
                                          Label  resourceLabel
                                          Form  namesAndClasses
                                                Toggle  dot
                                                Toggle  star
                                                Toggle  any
                                                Toggle  name
                                                Toggle  class
                                                     .
                                                     .
                                                     .
                                          Label  namesLabel
                                          List  namesList
                                          Label  constraintLabel
                                          List  constraintList
                                          Form  valueForm
                                                Label  valueLabel
                                                Text  valueText
```

```
                                    Box  commandBox
                                          Command  setFile
                                          Command  save
                                          Command  apply
                                          Command  saveAndApply
                                          Command  cancel
                                    Grip  grip
          Grip  grip
```

Environment

editres reads the following environmental variables:

DISPLAY The default host and display number.

XENVIRONMENT
The name of a resource file that overrides the global resources stored in the property **RESOURCE_MANAGER**.

Files

/usr/X11/lib/app-defaults/Editres— Required resources

See Also

bitmap, X utilities

Notes

This program is a prototype. *Caveat utilitor.*

editres was written by Chris D. Peterson, formerly of the MIT X Consortium.

ico — X Client

Animate an icosahedron or other polyhedron
ico [-display *name***] [-geometry** *geometry***] [-r] [-d** *pattern***] [-i] [-dbl] [-faces] [-noedges]**
[-sleep *n***] [-obj** *object***] [-objhelp] [-colors** *list***]**

ico displays and rotates a wire-frame or solid polyhedron. The object is either wire frame with the hidden lines removed, or solid with the hidden faces removed. A number of polyhedra are available; adding a new polyhedron to the program is quite simple.

ico recognizes the following command-line options:

-colors *color color ...*
Name the colors to use to draw the filled faces of the object. If fewer colors are named than the object has faces, **ico** reuses colors.

-d *pattern* Use *pattern* as a bit pattern for drawing the dashed lines of a wire-frame object.

-dbl Use double buffering on the display. This works for either wire-frame or solid-fill drawings. For solid-fill drawings, using this switch results in substantially smoother movement. Note that this option doubles the number of bit planes required. Because some colors typically are allocated by other programs, most eight-bit-plane displays will be limited to eight colors when using double buffering.

-faces Draw filled faces instead of wire frames.

-i Use inverted colors for wire frames.

-noedges Do not draw the wire frames. Typically, this option is used only when **-faces** is specified.

-r Dislay the polyhedron on the root window, instead of creating a new window.

-obj *object* Draw *object*. If no object is specified, **ico** draws an icosahedron.

-objhelp Print a list of the available objects, plus information about each object.

-sleep *n* Sleep *n* seconds between each movement of the object.

Adding Polyhedra

If you have the source code to **ico**, it is very easy to add more polyhedra. Each polyhedron is defined in a header file file of the name of **obj***XXX***.h**, where *XXX* relates to the name of the polyhedron. The format of the header file is defined in the file **polyinfo.h**. Look at the file **objcube.h** to see what the exact format of an **obj***XXX***.h** file should be, then model your new object's header file after that.

After making the new header file (or copying in a new one from elsewhere), simply invoke the command **make depend**. This recreates the file **allobjs.h**, which lists all of the header files. Executing **make** after this rebuild **ico** with the new object.

See Also

X clients

Notes

ico consumes an enormous number of computation cycles — so many, in fact, that you may have trouble getting the attention of the system in order to quit **ico**. Unless you have a very robust system, expect **ico** to "nail it to the wall". *Caveat utilitor.*

ico does not correctly display pyramids and tetrahedrons with filled faces.

imake — X Utility

C preprocessor interface to the make utility
imake [-D*define***] [-I***dir***] [-T***template***] [-f** *filename***] [-s** *filename***] [-e] [-v]**

The utility **imake** generates a **Makefile** from a template, a set of **cpp** macros, and a per-directory input file called **Imakefile**. This allows you to keep machine-dependencies (such has compiler options, alternate command names, and special **make** rules) apart from the descriptions of the items to be built.

Note that **imake** is almost always invoked through the script **/usr/X11/bin/xmkmf**. This script passes to **imake** the names of the appropriate configuration files needed to build a **Makefile** correctly under X Windows for COHERENT. The following information is included for the sake of completeness, but you probably will never need to use it.

Command-line Options

imake recognizes the following command-line options:

-D*define* Pass *define* directly to the C preprocessor **cpp**. This option typically is used to set directory-specific variables. For example, the X Window System uses this flag to set **TOPDIR** to the name of the directory that contains the top of the core distribution, and **CURDIR** to the name of the current directory, relative to the top.

-e Execute the generated **Makefile**. The default is let you do this later, by hand.

-f *filename* Name the per-directory input file. The default is **Imakefile**.

-I*directory* Pass *directory* directly to the C preprocessor **cpp**. This option typically is used to name the directory in which the **imake** template and configuration files may be found.

-s *filename* Name the **make** description file to be generated, but do not invoke **make**. The file name '-' indicates the standard output. The default is to generate, but not execute, a **Makefile**.

-T*template* Specify the name of the master template file used by **cpp**. This file usually is kept in a directory that is named with the option **-I**.

-v Verbose: print the **cpp** command line that **imake** uses to generate the **Makefile**.

How It Works

imake invokes **cpp** with any **-I** or **-D** flags that you passed to it on its command line, and passes it the following three lines:

```
#define IMAKE_TEMPLATE "Imake.tmpl"
#define INCLUDE_IMAKEFILE "Imakefile"
#include IMAKE_TEMPLATE
```

where **Imake.tmpl** and **Imakefile** may be overridden by its command-line options **-T** and **-f**, respectively

The **IMAKE_TEMPLATE** typically reads a file that contains machine-dependent parameters (specified as **cpp** symbols), a file of site-specific parameters, a file that defines variables, a file that contains **cpp** macro functions that generate **make** rules, and finally the **Imakefile** (specified by **INCLUDE_IMAKEFILE**) in the current directory. The **Imakefile** uses the macro functions to set the targets to build; **imake** generates the appropriate rules.

imake configuration files contain two types of variables: **imake** variables and **make** variables. **cpp** interprets the former when you run **imake**. By convention, they are in mixed case. The latter are copied into the **Makefile** for later interpretation by **make**. By convention, **make** variables are in upper case.

The rules file (usually named **Imake.rules** in the configuration directory) contains a variety of **cpp** macros that are configured for the machine on which the **Makefile** is being generated — in this case, X Windows for COHERENT. **Imake** replaces every occurrence of the string "@@" with a newline to allow macros that generate more than one line of **make** rules. For example, the macro

```
#define  program_target(program, objlist)                    @@\
program: objlist                                             @@\
         $(CC)  -o  $@  objlist  $(LDFLAGS)
```

when called with with the macro

```
program_target(foo, foo1.o  foo2.o)
```

expands into:

```
foo:  foo1.o  foo2.o
      $(CC)  -o  $@  foo1.o  foo2.o  $(LDFLAGS)
```

On systems whose **cpp** reduces multiple tabs and spaces to one space character, **imake** attempts to replace any necessary tab characters. (**make** is very picky about the difference between tabs and spaces). For this reason, you must precede every colon ':' in a command line with a backslash '\'.

Use With the X Window System

The X Window System uses **imake** extensively to build both the source tree and external software. As mentioned above, **imake** sets two special variables, **TOPDIR** and **CURDIR**, to ease the referencing files via relative path names. For example, **imake** automatically generates the following command to build the **Makefile** in the directory **lib/X11** (relative to the top of the sources):

```
%  ../.././config/imake  -I../.././config  \
      -DTOPDIR=../../.   -DCURDIR=./lib/X11
```

When building X programs outside the source tree, define the special symbol **IUseInstalled** and omit **TOPDIR** and **CURDIR**.

In most cases, you should use the script **xmkmf**, rather than attempt to do this by hand.

Input Files

The following names the files that **imake** reads. The indentation shows which files are included by other files:

Imake.tmpl (*generic variables*)
- site.def (*site-specific,* **BeforeVendorCF** *defined*)
- *.cf (*machine-specific*)
  - *Lib.rules (*shared-library rules*)
- site.def (*site-specific,* **AfterVendorCF** *defined*)
- Project.tmpl (*X-specific variables*)
  - *Lib.tmpl (*shared-library variables***)**
- **Imake.rules (***rules*)

Imakefile
- Library.tmpl (*library rules*)
- Server.tmpl (*server rules*)

All of these are kept in directory **/usr/X11/lib**.

Note that **site.def** is included twice: once before the ***.cf** file and once after. Although most site customizations should be specified after the ***.cf** file, some, such as the choice of compiler, need to be specified before, because other variable settings may depend on them.

LEXICON

The first time **site.def** is included, the variable **BeforeVendorCF** is defined; the second time, the variable **AfterVendorCF** is defined. All code in **site.def** should be inside an **#ifdef** for one of these symbols.

Environment Variables

imake reads the following environmental variables. Note, however, that their use is *not* recommended as they introduce dependencies that are not readily apparent when you run **imake**:

IMAKEINCLUDE
This should be a valid include argument for the C preprocessor; e.g., **-I/usr/include/local**. Actually, any valid **cpp** argument will work here.

IMAKECPP
This should be a valid path to a preprocessor program; e.g., **/usr/local/cpp**. By default, **imake** uses **/lib/cpp**.

IMAKEMAKE
This should be a valid path to a **make** program, such as **/usr/local/make**. By default, **imake** uses whatever **make** program is found using **execvp()**. **imake** uses this variable only if its command-line option **-e** is specified.

Example

As can be seen from the above descriptions, **imake** is a complex program, and the writing of an **Imakefile** a difficult exercise. However, one appealing feature of **imake** is that if you adhere to the default settings of the local implementation, an **Imakefile** can be quite simple.

For example, the following gives the **Imakefile** for the program **xwave**, whose source code is included with X Windows for COHERENT:

```
LOCAL_LIBRARIES = $(XLIB)

OBJS = xwave.o force.o plot.o prop.o

SRCS = xwave.c force.c plot.c prop.c

SYS_LIBRARIES = -lm

ComplexProgramTarget(xwave)
```

LOCAL_LIBRARIES expands into the standard suite of X libraries. **SYS_LIBRARIES** gives the COHERENT libraries that the program also requires; in this case, **xwave** also requires the mathmatics library **libm.a**.

OBJS names the relocatable object modules from which the executable is built. **SRCS** names the source files from which the objects are built. There is usually — though by no means always — a one-to-one correspondence between source files and objects. Most source files consists of files of C code, although some may be written in other languages (e.g., C++, **yacc**, or **lex**), or be in some specialized format (e.g., uuencoded).

Finally, the function-like macro **ComplexProgramTarget()** expands into a mass of instructions that will build **xwave**.

To convert this **Imakefile** into a **Makefile**, simply enter the directory that holds it and type the command **xmkmf**. This five-line **Imakefile** expands into several hundred lines of **Makefile** that will build **xwave**.

Files

/usr/tmp/tmp-imake.*nnnnnn* — Temporary input file for **cpp**
/usr/tmp/tmp-make.*nnnnnn* — Temporary input file for **make**
/lib/cpp — Default C preprocessor

See Also

xmkmf, X utilities
COHERENT Lexicon: **C preprocessor, cpp, make**
COHERENT tutorial: *The make Programming Discipline*

Notes

Because **imake** invokes the C preprocessor **cpp** to expand macros, its behavior will vary from system to system depending upon the C preprocessor used: an **Imakefile** that compiles correctly on one system may die a painful death on another because of differences in the preprocessor. Please note the following common problems that are seen under COHERENT:

0.3i **imake** normally recognizes a line that begins with a '#' as being a comment. However, if a comment contains an apostrophe, **imake** will pass that line to **cpp**, which will then attempt to interpret it as a C preprocessor instruction. The solution is to remove the comment, or to surround it with C-style comments.

- Some **Imakefile**s that are built from a resource-control system (RCS) will have an RCS-related comment at the top. This comment will also confuse **imake**. Again, the solution is to remove the comment, or to enclose it within C-style comments.

imake was written by Todd Brunhoff of Tektronix and MIT Project Athena, and Jim Fulton of the MIT X Consortium.

imake requires the C preprocessor from the GCC compiler. This is included with X Windows for COHERENT.

Makefiles generated by **imake** must be processed through **gmake**, not the default COHERENT implementation of **make**.

listres — X utility

List resources in widgets
listres [-*option***]**

listres generates a listing of a widget's resource data base. The listing gives the class in which each resource is first defined, the instance and class name, and the type of each resource. **listres** generates a listing for every widget named on its command line. If the command line names no widgets, or includes the switch **-all**, **listres** generates a listing for every widget.

listres recognizes the following command-line options:

-all Print information for all known widgets and objects.

-format *printf-string*
Use a **printf()**-style format string to print the name, instance, class, and type of each resource.

-nosuper Do not list resources that are inherited from a superclass. This is useful for determining which resources are new to a subclass.

-top *name* *name* gives the widget to be treated as the top of the hierarchy. Case is not significant, and *name* can match either the class variable name or the class name. The default is **core**.

-variable Identify widgets by the names of the class-record variables rather than by the class name given in the variable. This is useful for distinguishing subclasses that have the same class name as their superclasses.

See Also

editres, X utilities
COHERENT Lexicon: **printf()**

Notes

listres was written by Jim Fulton of the MIT X Consortium.

maze — X Client

Create and solve a random maze
maze [-S] [-r] [-geometry *geometry***] [-display** *display***]**

maze creates a "random" maze, and then solves it on its own. It recognizes the following command-line options:

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on *host*.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-r Simulate reverse video by exchanging the foreground and background colors.

-S Full-screen window: Fill the screen with the **maze** window.

Clicking the left mouse button clears the window and restarts **maze**. Clicking the middle mouse button toggles the program: the first click tells **maze** to stop, and the second click tells it to continue. Finally, clicking the right mouse button kills **maze**.

See Also

puzzle, X clients, xgas, xtetris

Notes

maze insists on playing by itself — you cannot try to solve the maze on your own.

maze was written by Richard Hess of Consilium, and Dave Lemke and Martin Weiss of Sun MicroSystems.

mkdirhier — X Utility

Make a directory hierarchy
mkdirhier *directory ...*

The X utility **mkdirhier** creates each *directory*. Unlike the COHERENT command **mkdir**, **mkdirhier** creates any of the parent directories of *directory* if do not exist.

See Also

X utilities
COHERENT Lexicon: **mkdir**

mkfontdir — X Utility

Create file fonts.dir from directory of font files
mkfontdir [*directory ...***]**

The X utility **mkfontdir** constructs a table of font names. For every *directory*, it reads every font file and constructs a font name for the font in that file. **mkfontdir** derives the font name from the property **FONT** within the font file, or, if the file contains no such property, from name of the font file stripped of its suffix. **mkfontdir** writes the name of each font and the file that contains into file **fonts.dir** in *directory*.

The kinds of font files that **mkfontdir** reads depends on configuration parameters, but typically include PCF fonts (suffix, **.pcf**), SNF fonts (suffix, **.snf**), and BDF fonts (suffix, **.bdf**). If a font exists in multiple formats, **mkfontdir** first chooses PCF, then SNF, and finally BDF.

Scalable Fonts

Because scalable-font files do not usually include the X font name, you must edit by hand the file **fonts.dir** in a directory that contains such fonts, to include the appropriate entries for these fonts. However, when you run **mkfontdir**, it will erase all of those additions, so be careful.

Font Name Aliases

Both the X server and the font server look for files **fonts.dir** and **fonts.alias** in each directory in the font path each time it is set.

The file **fonts.alias**, which can be put in any directory of the font path, maps new names to existing fonts, and should be edited by hand. The format is straightforward: two columns separated by white space, first of which contains aliases and the second of which contains font-name patterns.

When you use a font alias, the X server looks through each font directory in turn, and searches in the normal manner for the name the alias references. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white-space in either name, simply enclose them in quotation marks. To embed a literal quotation mark (or any other character), precede it with backslash '\'. For example:

```
"magic-alias with spaces"     "\"font\name\" with quotes"
regular-alias                 fixed
```

If the string **FILE_NAMES_ALIASES** stands alone on a line, each file-name in the directory (stripped of its suffix) is used as an alias for that font.

See Also

fs, xset, X utilities

oclock — X Client

Display an analogue clock
oclock [*options***]**

oclock is an X client that displays an analogue clock on the screen:

The clock's time is set to the current system time, and is continually updated to show the current time.

oclock recognizes the following options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the clock on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-geometry *geometry*
Set the geometry of the clock to *geometry*. The term "geometry" means the dimensions of the clock and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-hour *color* Set the color of the hour hand to *color*.

-jewel *color* Set the color of the "jewel" (the diamond-shaped spot that marks 12 o'clock) to *color*.

-minute *color*
Set the color of the minute hand to *color*.

-noshape Forbid the clock to reshape itself.

Colors

If you want your clock to be viewable in color, include the following in the **#ifdef COLOR** section you read with **xrdb**:

```
*customization:                -color
```

This causes **oclock** to pick up the colors in the color-customization file **/usr/X11/lib/app-defaults/Clock-color**. Below are the default colors:

```
Clock*Background: grey
Clock*BorderColor: light blue
Clock*hour: yellow
Clock*jewel: yellow
Clock*minute: yellow
```

See Also

X clients, xclock

Notes

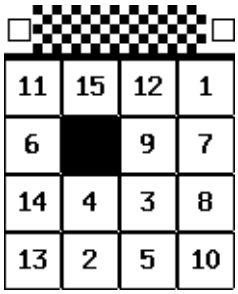
Unlike the X client **xclock**, **oclock** uses a round window.

oclock was written by Keith Packard of the MIT X Consortium.

puzzle — X Client

The X scrambled-number game
puzzle [-*option ...***]**

The X client **puzzle** implements one of those maddening scrambled-number puzzles. The puzzle is divided into 16 squares: one is blank, and the other 15 are numbered, one through 15. The point of the game is to unscramble the numbered squares, so they are in numeric order, from top to bottom and left to right, as follows:



Clicking the small, square button at the top left of the puzzle scrambles the squares into random order. To move a numbered square, click on it; the square slides into the empty square. When you give up trying to solve the puzzle, click the small, square button at the top right of the puzzle; **puzzle** then unscrambles the puzzle automatically — just to prove how much smarter it is than the average user.

puzzle recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-picture *file*
Build the puzzle from the image in *file*, instead of numbered tiles. *file* must be the special **puzzle** picture format. The X client **xgrabsc** (which is not included in this package) can "grab" images of windows and store them in the **puzzle** format, for use with **puzzle**. Note that there is a strict limit on the size of image that **puzzle** can use.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

See Also

maze, X clients, xgas, xtetris

resize — X Utility

Set environmental variables to show window size
resize [-cu]

The X utility **resize** generates a shell script that resets the environmental variables **COLUMNS** and **ROWS** to reflect the dimensions of the window within which it was run.

resize recognizes the following options:

-c Generate a script for the C shell, regardless of the shell you are using. Note that COHERENT does include the C shell, although one can be obtained from a third-party source.

-u Generate a script for the Bourne shell, regardless of the shell you are using.

See Also

X utilities, xterm

Notes

Most COHERENT screen-oriented applications assume that the size of the screen is 24 rows by 80 columns. This will change gradually in the future; but at present, do not expect this command do have much effect on the way things work.

This client is by Mark Vandevoorte of MIT Project Athena and Edward Moy of the University of California, Berkeley.

showrgb — X Utility

Un-compile an RGB color-name data base
showrgb [*database* **]**

showrgb reads an RGB color-name data base compiled for use with the DBM data-base routines, converts it back to source form, and prints the result onto standard output. The default data base is the one that X was built with, and may be overridden on the command line. Specify the data base name without the suffix **.pag** or **.dir**.

Files

/usr/X11/lib/rgb.dir — Default data base.

See Also

xcmsdb, xstdcmap, X utilities

startx — X Utility

Initiate an X session
/usr/X11/bin/startx [[*client* **]** *options ...* **] [-- [** *server* **]** *options ...* **]**

The script **startx** is a front end to the application **xinit**. It provides a somewhat easier interface for running a session of the X Window System. It typically is run with no arguments.

To determine which client to run, **startx** first looks for a file called **$HOME/.xinitrc**. If it cannot find that file, it uses the file **/usr/X11/lib/xinit/xinitrc**. If command-line client options are given, they override this behavior.

$HOME/.xinitrc typically is a shell script that starts many clients according to the user's preference. When this shell script exits, **startx** kills the server and performs any other tasks needed to shut down a session. Most of the clients that **$HOME/.xinitrc** starts should be run in the background. The last client should run in the foreground; when it exits, the session exits. People often choose a session manager, window manager, or **xterm** as the "magic" client.

Example

Below is a sample **$HOME/.xinitrc** that starts several applications and leaves the window manager running as the "last" application. Assuming that the window manager has been configured properly, the user then chooses the **Exit** menu item to shut down X.

```
xrdb  -load  $HOME/.Xresources
xsetroot  -solid  gray  &
xbiff  -geometry  -430+5  &
oclock  -geometry  75x75-0-0  &
xterm  -geometry  +0+60  -ls  &
xterm  -geometry  +0-100  &
xconsole  -geometry  -0+0  -fn  5x7  &
twm
```

Environment

startx sets the environmental variable **DISPLAY**, which names the display to which clients should connect.

Files

$(HOME)/.xinitrc — Clients to run
$(HOME)/.xserverrc — Server to run; the default is **X**
/usr/X11/lib/xinit/xinitrc — Name default clients
/usr/X11/lib/xinit/xserverrc — Name default server

See Also

xinit, X utilities

twm — X Utility

Tab Window Manager for the X Window System
twm [-display *dpy***] [-s] [-f** *initfile***] [-v]**

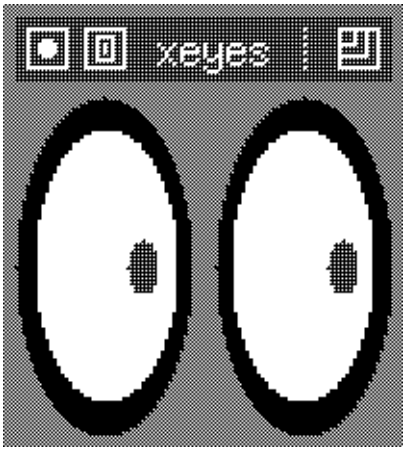
twm is a window manager for the X Window System. It provides title bars, shaped windows, several forms of icon management, user-defined macro functions, click-to-type and mouse-driven keyboard focus, and user-specified bindings for keys and mouse buttons.

The following describes how to "program" **twm** — that is, how to modify its appearance and its behavior by setting variables, and by defining functions and macros. For a description of how to use **twm** interactively, see the tutorial at the beginning of this manual. This tutorial also introduces many of the terms used in this article, and describes **twm**'s in the grand hierarchy of the X Window System.

Invoking twm

To begin, **twm** usually is started by the X utility **xinit**. When launched by **xinit**, **twm** frequently is executed in the foreground as the last client named in the file **$HOME/.xinitrc**. When run this way, exiting **twm** terminates the X session.

When **twm** opens a window for an application, it surrounds the window with a "frame" that has special border and a title bar at the top. The following shows the window for the X client **xeyes**:



The top of the window has a *title bar* , which displays the name of the program, and three screen buttons. The following describes the buttons from left to right:

- The leftmost button, which has a bullet in it, "iconifies" the program: when you click it, the window disappears from the screen and its icon at the top of the screen displays the **X** logo. To redraw the window on the screen, click on its icon.
- The second button from the left, which displays displays a stylized capital 'D', controls a menu for modifying aspects of the window. When you click on this button, a drop-down menu appears that has the following entries:

  `(Iconify)`
  Iconify the window. This is identical to clicking on the leftmost button.

  `(Lower)` If this window overlaps with one or more other windows, this option puts this window in the background below the overlapping windows.

  `(Move)` Move the window. An outline of the window appears, which you can drag to another position on the screen. When you release the left mouse button, **twm** re-draws the window in its new position.

  `(Raise)` If this window overlaps with one or more other windows, raise this to the foreground atop the other windows.

  `(Refresh)`
  Re-draw the window, to eliminate "garbage" that may have appeared on the screen.

  `(Resize)`
  Resize the window: **twm** draws an outline of the window; by dragging the mouse, you can change the size of the window. Note that the size of the window does not change until the you drag the mouse cursor past the current border of the window; therefore, if you want to make the window smaller, first stretch the window larger, then shrink it.

  `(Applications)`
  Display the Applications menu, the same menu you see if you click the left mouse button while the mouse cursor is not in any application window.

  `(Properties)`
  Display a sub-menu that has three entries: `(Autoraise)`, `(Focus)` and `(Unfocus)`. Selecting an entry from the sub-menu toggles the corresponding property of the window.

  Twm Operations
  Display the Operations menu, the same menu you see if you click the right mouse button while the mouse cursor is not in any application window.

  `(Info)` Display a pop-up window that gives the geometry of the window, plus other information.

  `(Kill_Program)`
  Invoke the **twm** function **f.destroy**, which kills an application. This function is described below.

  `(Close_Window)`
  Invoke the **twm** function **f.delete**, which closes the current window. If you close all of an application's windows, the application dies gracefully. This function, which is less violent than **f.destroy**, is described below.
- The rightmost button, which displays four nested boxes, lets you move the window.

In addition, clicking on the title bar itself, apart from the three buttons, lets you move the window.

For details on how to modify the appearance of the title bar, see below for the description of the variable **ShowIconManager** and of the function **f.showiconmgr**.

When a client creates a new window, **twm** honors any size and location information, usually requested through the command-line option **-geometry**, or the application's resources. If the user supplies no location or size when he invokes the application, **twm** displays an outline of the window's default size, its title bar, and lines that divide the window into a 3×3 grid that tracks the mouse cursor. Clicking the left mouse button fixes the window at the current position and give it the default size. Pressing the middle mouse button (or both mouse buttons, should the mouse have only two buttons) and dragging the outline gives the window its current position but lets you resize the sides as described above. Clicking the right mouse button tells **twm** to give the window its current position but make it long enough to touch the bottom the screen.

Options

twm recognizes the following command-line options:

-display *dpy*
Specify the display to use. Under COHERENT X, **twm** recognizes only the default display.

-f *filename*
Name the start-up file to use. By default, **twm** uses file **$HOME/.twmrc**. If no such file exists, it uses file **/usr/X11/lib/twm/start.twmrc**.

-s Manage only the default display, as specified by option **-display** or by the environmental variable **DISPLAY**. By default, **twm** attempts to manage all displays under a given server.

-v Verbose: tell **twm** to print error messages whenever an unexpected X error occurs. This can be useful when debugging an applications, but can be distracting in regular use.

Customization

When **twm** comes up, it reads the following start-up files, in the order in which the appear here:

$HOME/.twmrc
This gives your personal start-up file on systems that have only one screen.

/usr/X11/lib/twm/system.twmrc
If it does not find preceding file, **twm** reads this file for a default configuration. You can tailor this file to give novice users a default configuration.

If you want to customize **twm** to suit your tastes, copy file **/usr/X11/lib/twm/system.twmrc** into your home directory, rename it **.twmrc**, and modify it as you like. You can, for example, change the default colors used on the windows, add or delete entries in **twm**'s pop-up menus, or change the options on the programs that the menus invoke. The tutorial *Using X Windows*, earlier in this manual, introduces how to modify this file.

If it finds neither of these start-up files, **twm** uses the built-in defaults described above. The only resource used by **twm** is **bitmapFilePath**, which gives a colon-separated list of directories to search when looking for bit-map files.

A start-up file contains the following classes of specifications:

Variables
These specifications come first in a start-up file. They describe fonts, colors, cursors, border widths, icon and window placement, highlighting, autoraising, layout of titles, jumping the cursor (or "warping"), and use of the icon manager.

Bindings
These specifications usually come in the start-up file after the **Variables** specifications. They specify the functions to be invoked when keyboard and mouse buttons are pressed in windows, icons, titles, and frames.

Menus These specifications give any user-defined menus. These menus contain functions to be invoked or commands to be executed.

Variable names and keywords are case-insensitive. Strings must be surrounded by quotation marks, and are case-sensitive. A pound sign '#' outside of a string tells **twm** to treat the rest of the line as a comment.

The following three sections detail these classes of specifications.

Variables

Many aspects of **twm**'s user interface are controlled by variables that you can set at the beginning of a start-up file. Some variables are Boolean — that is, the option a variable controls is turned on if that variable is present, and is turned off if it is absent. Other options require keywords, numbers, strings, or lists of all of these.

Some variables can take a list of options or arguments that are enclosed within braces; items within a list are separated by white space or newlines. For example:

```
AutoRaise { "xclock" "XTerm" }
```

or

```
AutoRaise
{
      "xclock"
      "XTerm"
}
```

When **twm** searches a variable that contains a list of strings that represent windows (e.g., to determine whether to enable **AutoRaise**, as shown above), the string must exactly match the window's name (given by the window property **WM_NAME**), or its resource or class name (both given by the window property **WM_CLASS**). The above example turns on **AutoRaise** for all windows named **xclock** and all windows of class **XTerm**.

A string argument that is interpreted as a file name (as with the resources **Pixmaps**, **Cursors**, and **IconDirectory**) is prefixed by your home directory (as given by the environmental variable **HOME**) if its first character is a tilde '~'. If, however, its first character is a colon ':', **twm** assumes that the name refers to one of the internal bit-maps used to create the default title bars symbols, that is, **:xlogo** or **:iconify** (both hold the X logo used within the `(iconify)` button), **:resize** (the nested squares used within the `(resize)`button), and **:question** (the question mark used in place of non-existent bit-map files).

You can set the following variables at the beginning of a start-up file:

AutoRaise { *win-list* }
: Raise every window automatically whenever the mouse cursor enters it. This action can be enabled or disabled interactively on an individual window by using the function **f.autoraise**.

AutoRelativeResize
: When you size or resize a window, **twm** by default waits for the mouse cursor to cross an edge of the window before it begins the resizing operation. Command **AutoRelativeResize** turns off this default behavior. Instead, moving the mouse cursor automatically shifts the nearest edge or edges by the same amount. This lets you resize windows that extend past the edge of the screen. This option is particularly useful for people who like the press-drag-release method of sweeping out the size of a window.

BorderColor *string* **[{** *wincolorlist* **}]**
: Set the default color of the border drawn around every non-iconified window. This variable can be given only within a **Color** or **Monochrome** list. The optional *wincolorlist* lists pairs of windows and colors, with the given window being assigned the corresponding color. For example:

```
BorderColor "gray50"
{
      "XTerm"     "red"
      "xclock"    "green"
}
```

  The default border color is **black**.

BorderTileBackground *string* **[{** *wincolorlist* **}]**
: Set the default background color in the gray pattern used in unhighlighted borders. It is used only if **NoHighlight** has not been set, and it can be given only within a **Color** or **Monochrome** list. *wincolorlist* sets the color for specific windows, as described above for **BorderColor**. The default color is **white**.

BorderTileForeground *string* **[{** *wincolorlist* **}]**
: Exactly like **BorderTileBackground**, described above, except that it sets the foreground color rather than the background color. The default color is **black**.

BorderWidth *pixels*
: Set the width, in pixels, of the border that surrounds a window frame. This variable applies to every client has not set the variable **ClientBorderWidth**, and to windows that **twm** creates (e.g., the icon manager). The default width is two pixels.

ButtonIndent *pixels*
: Set the amount, in pixels, by which title buttons are indented on all sides. Positive values make the buttons smaller than the window text and highlight area, so they stand out. Setting this and the variable **TitleButtonBorderWidth** to zero makes title buttons as tall and wide as possible. The default is one pixel.

ClientBorderWidth
: Set the width of the border of a window's frame to the width of the window's initial border, rather than to the value of **BorderWidth**.

Color { *colors-list* }
Set the colors to use on polychromatic displays. *colors-list* consists of the following color variables and their values:

DefaultBackground
DefaultForeground
MenuBackground
MenuForeground
MenuTitleBackground
MenuTitleForeground
MenuShadowColor

Each of the following color variables can take a list of pairs of window names and colors, thus letting you assign specific colors to specific windows. For details, see the description of the variable **BorderColor**, above:

BorderColor
IconManagerHighlight
BorderTitleBackground
BorderTitleForeground
TitleBackground
TitleForeground
IconBackground
IconForeground
IconBorderColor
IconManagerBackground
IconManagerForeground

For example:

```
Color
{
        MenuBackground          "gray50"
        MenuForeground          "blue"
        BorderColor             "red" { "XTerm" "yellow" }
        TitleForeground         "yellow"
        TitleBackground         "blue"
}
```

All of these color variables may also be specified for the variable **Monochrome**, thus letting you use the same initialization file for both color and monochrome displays.

For a list of recognized colors, see the file **/usr/X11/lib/rgb.txt**.

ConstrainedMoveTime *milliseconds*
Set the time interval during which two button clicks begin a constrained-move operation. Double-clicking within this period of time when invoking **f.move** causes the window to move only horizontally or vertically. Setting this variable to zero disables constrained moves. The default is 400 milliseconds.

Cursors { *cursor-list* }
Set the glyphs that **twm** uses for mouse cursors. Each cursor can be defined either from the **cursor** font or from two bit-map files. Shapes from the **cursor** font can be specified directly, as follows:

cursorname "*string*"

where *cursorname* is one of the cursor names given below, and *string* is the name of a glyph, as set in the file **/usr/X11/include/cursorfont.h** (without the prefix **XC**_).

If the cursor is to be defined from a bit-map file, use the following syntax:

cursorname "*image*" "*mask*"

image and *mask* name the files that contain, respectively, the glyph image and mask in bit-map form. (For details on bit-mapped images, see the Lexicon entry for the X client **bitmap**). The cursor bit-map files are located in the same manner as icon bit-map files.

The following gives the default cursor definitions:

```
Cursors
{
        Frame                "top_left_arrow"
        Title                "top_left_arrow"
        Icon                 "top_left_arrow"
        IconMgr              "top_left_arrow"
        Move                 "fleur"
        Resize               "fleur"
        Menu                 "sb_left_arrow"
        Button               "hand2"
        Wait                 "watch"
        Select               "dot"
        Destroy              "pirate"
}
```

For a table that shows available cursor shapes, see the entry in this manual for the X program **xfd**.

DecorateTransients
Give title bars to transient windows (that is, windows that contain the property **WM_TRANSIENT_FOR**). By default, transient windows are not reparented.

DefaultBackground *string*
Set the background color used in the sizing and information windows. The default is **white**.

DefaultForeground *string*
Set the foreground color used in the sizing and information windows. The default is **black**.

DontIconifyByUnmapping { *win-list* }
List the windows not to be iconified by simply unmapping the window (as is the case if the variable **IconifyByUnmapping** has been set). This variable often is used to force some windows to be treated as icons whereas other windows are handled by the icon manager.

DontMoveOff
Do not let windows be moved off the screen. The function **f.forcemove** overrides this function.

DontSqueezeTitle [{ *win-list* }]
Do not squeeze title bars, as described below for the variable **SqueezeTitle**. If the optional list of windows is supplied, only the windows named therein are stopped from being squeezed.

ForceIcons
The icon pixel maps specified in the variable **Icons**, described below, should override any client-supplied pixel maps.

FramePadding *pixels*
Set the gap between the title-bar decorations (the button and text) and the window's frame. The default is two pixels.

IconBackground *string* [{ *win-list* }]
Set the background color of icons. It can be specified only within a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **white**.

IconBorderColor *string* [{ *win-list* }]
Set the color of the border of icon windows. It can be specified only inside of a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **black**.

IconBorderWidth *pixels*
Set the width, in pixels, of the border of icon windows. The default is two pixels.

IconDirectory *string*
Set the directory that **twm** should search if it cannot find a bit-map file in any of the directories named in the resource **bitmapFilePath**.

IconFont *string*
Set the font to be used to display icon names within icons. The default font is **variable**.

IconForeground *string* **[{** *win-list* **}]**
Set the foreground color used when displaying icons. It can be specified only inside of a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **black**.

IconifyByUnmapping [{ *win-list* **}]**
Iconify windows by unmapping them, without trying to map any icons. This assumes that you will remap the window through the icon manager, the function **f.warpto**, or the menu **TwmWindows**. If the optional *win-list* is provided, only the windows it names will be iconified by unmapping. Windows that have set both this and the option **IconManagerDontShow** may not be accessible if the user's start-up file does not contain a binding to the menu **TwmWindows**.

IconManagerBackground *string* **[{** *win-list* **}]**
Set the background color for icon-manager entries. It can be specified only within a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **white**.

IconManagerDontShow [{ *win-list* **}]**
The icon manager should not display any windows. If the optional *win-list* is given, only the windows named therein will not be displayed. This variable is used to prevent windows that are rarely iconified (such as **xclock** or **xload**) from taking up space in the icon manager. For example:

```
IconManagerDontShow
{
    "Virtual Desktop"
    "xbiff"
    "xclock"
    "xload"
    "oclock"
}
```

IconManagerFont *string*
Set the font used when displaying icon-manager entries. The default font is **variable**.

IconManagerForeground *string* **[{** *win-list* **}]**
Set the foreground color used when displaying icon-manager entries. It can be specified only within a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **black**.

IconManagerGeometry *string* **[** *columns* **]**
Set the geometry of the icon-manager window. *string* gives the dimensions of the icon manager's window, in standard nomenclature. **twm** divides that window into *columns* pieces, which are scaled according to the number of entries in the icon manager. The icon manager arranges its icons in rows, each of which contains *columns* icons. The default number of columns is one.

IconManagerHighlight *string* **[{** *win-list* **}]**
Set the border color to be used when highlighting the icon manager entry that currently has the focus. It can be set only inside of a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **black**.

IconManagers { *iconmgr-list* **}**
This variable specifies a list of icon managers to create. Each item in *iconmgr-list* has the following format:

"*winname***" ["***iconname***"] "***geometry***"** *columns*

where *winname* names the window to put into this icon manager, *iconname* names the icon manager window's icon, *geometry* is a standard geometry specification, and *columns* is the number of columns in this icon manager as described by the variable **IconManagerGeometry** (described above). For example:

```
IconManagers
{
      "XTerm"     "=300x5+800+5"    5
      "myhost"    "=400x5+100+5"    2
}
```

In this example, a client whose name or class is **XTerm** will has an entry created in the **XTerm** icon manager, and a client whose name is **myhost** will be put into the **myhost** icon manager.

IconManagerShow { *win-list* }
The windows in *win-list* appear in the icon manager. When used with the variable **IconManagerDontShow**, only the windows in this list appear in the icon manager.

IconRegion *geomstring vgrav hgrav gridwidth gridheight*
Set the area in the root window to place icons if the client provides no icon-location information. *geomstring* is a quoted string that contains a standard geometry specification. If multiple *IconRegion* lines are given, **twm** places icons put into succeeding icon regions when the first is full. *vgrav* controls whether icons fill the region from top to bottom or vice versa; this must be either **North** or **South**. Likewise, *hgrav* controls whether icons fill the region from left to right or vice versa; it must be either **East** or **West**. Icons are laid out within the region on a grid whose cells are *gridwidth* pixels wide and *gridheight* pixels high.

Icons { *win-list* }
Name the bit-map files to use as icons for selected windows. For example:

```
Icons
{
      "XTerm"  "xterm.icon"
      "xfd"    "xfd_icon"
}
```

Windows that match **XTerm** are not iconified by unmapping, and use the icon bit-map in file **xterm.icon**. If the variable **ForceIcons** is set, **twm** uses this bit-map even if the client has requested its own icon pixel map.

InterpolateMenuColors
Interpolate menu-entry colors between the specified colors. Consider the example:

```
Menu "mymenu"
{
        "Title"        ("black":"red")       f.title
        "entry1"                             f.nop
        "entry2"                             f.nop
        "entry3"       ("white":"green")     f.nop
        "entry4"                             f.nop
        "entry5"       ("red":"white")       f.nop
}
```

The foreground colors for **entry1** and **entry2** are interpolated between black and white, and the background colors between red and green. Likewise, the foreground color for **entry4** is half-way between white and red, and the background half-way between green and white.

MakeTitle { *win-list* }
List the windows on which a title bar should be placed. Use it to request titles on specific windows when **NoTitle** has been set.

MaxWindowSize *string*
Set the maximum size for a window. This typically is used to prevent a window from becoming larger than the screen. The default is **30000×30000**.

MenuBackground *string*
Set the background color for menus. It can be used only within a **Color** or **Monochrome** list. The default color is **white**.

MenuFont *string*
Set the font to use when displaying menus. The default is font **variable**.

MenuForeground *string*
Set the foreground color for menus. It can be used only within of a **Color** or **Monochrome** list. The default color is **black**.

MenuShadowColor *string*
Set the color of the shadow behind a pull-down menu. It can be used only within a **Color** or **Monochrome** list. The default color is **black**.

MenuTitleBackground *string*
Set the background color for **f.title** entries in a menu. It can be used only within a **Color** or **Monochrome** list. The default color is **white**.

MenuTitleForeground *string*
Set the foreground color for **f.title** entries in a menu. It can be used only within of a **Color** or **Monochrome** list. The default color is **black**.

Monochrome { *colors* }
List the color assignments to make if the screen is monochrome. See the description of the variable **Colors**, above.

MoveDelta *pixels*
Set the number of pixels the mouse cursor must move before the function **f.move** starts to work. Also, see the description of the function **f.deltastop**, below. The default is zero pixels.

NoBackingStore
Menus should not use backing storage. This minimizes repainting of menus. This is typically used with servers that can repaint faster than they can handle backing storage.

NoCaseSensitive
Ignore case when sorting icon names within an icon manager. This option is typically used with applications that capitalize the first letter of their icon name.

NoDefaults
Do not supply the default title buttons and bindings. Use this option only if the start-up file contains an entirely new set of bindings and definitions.

NoGrabServer
Do not grab the server when it pops up menus or moves opaque windows.

NoHighlight [{ *win-list* }**]**
Do not highlight the borders of a window when the mouse cursor enters it or its icon. If the optional *win-list* is given, **twm** disables highlighting only for the windows named therein. When the border is highlighted, **twm** draws it in the current **BorderColor**; when the border is not highlighted, **twm** stipples it with a gray pattern constructed from the variables **BorderTileForeground** and **BorderTileBackground**.

NoIconManagers
Do not use an icon manager.

NoMenuShadows
Do not draw drop shadows behind menus. Use this with slower servers, because it speeds up menu drawing at the expense of making the menu slightly harder to read.

NoRaiseOnDeiconify
Do not raise window to the foreground when the user de-iconifies it.

NoRaiseOnMove
Do not raise a window to the foreground when the user moves it. This lets windows slide underneath each other.

NoRaiseOnResize
Do not raise a window to the foreground when the user resizes it. This allow windows to be resized beneath each other.

NoRaiseOnWarp
Do not raise a window to the foreground when the function **f.warpto** jumps (or "warps") the mouse cursor into it. If this option is set, jumping to an occluded window may force in the mouse cursor into the occluding window instead the occluded window (which causes unexpected behavior with **f.warpring**).

NoSaveUnders
Do not request save-unders. This minimizes window repainting after menu selection. This variable typically is set on displays that can repaint faster than they can handle save-unders.

NoStackMode [{ *win-list* }**]**
Ignore requests from client windows to change stacking order. If the optional *win-list* is given, **twm** ignores requests only from the windows named therein. This typically is used to stop a "pushy" application from relentlessly popping itself into the window's foreground. Do not give title bars to windows. If the optional *win-list* is given, only the windows named therein should not have title bars. Use the variable **MakeTitle** with this option to force **twm** to draw title bars only in specific windows.

NoTitleFocus
Do not set the keyboard-input focus to a window when the mouse cursor enters it. Normally, **twm** sets the focus so that focus and key events from the title bar and icon managers are delivered to the application. If you move the mouse cursor quickly and **twm** is slow to respond, it may direct input into the old window instead of into the new. This option is typically used to prevent this "input lag" and to work around bugs in older applications that have problems with focus events.

NoTitleHighlight [{ *win-list* **}]**
Do not display the highlight area of the title bar; this area indicates the window that has the input focus. If the optional *win-list* is given, only the windows named therein are not highlighted. Setting this and the variable **SqueezeTitle** substantially reduces the amount of screen space required by title bars.

OpaqueMove
The function **f.move** moves the window, instead of just an outline of it. This option is typically used on fast displays (particularly if variable **NoGrabServer** is set).

Pixmaps { *pixmaps* **}**
List the pixel maps that define the appearance of various images. Each entry consists of two strings: the first names the pixel map to set, and the second names the bit-map file. The following pixel maps may be specified:

```
Pixmaps
{
      TitleHighlight    "gray1"
}
```

The default for **TitleHighlight** is an even stipple pattern.

RandomPlacement
When **twm** opens an application whose window has no specified geometry, it should drop the window into a pseudo-random location instead of having the user drag an outline of the window to where she wants it.

ResizeFont *string*
Name the font to display in the dimensions window as **twm** resizes a window. The default font is **fixed**.

RestartPreviousState
Use the property **WM_STATE** on client windows to remember which windows should be iconified and which should be left visible. This typically is used to regenerate the state the screen was in before the previous window manager was shut down.

SaveColor { *colors-list* **}**
List the color assignments to be stored as pixel values in the root-window property **_MIT_PRIORITY_COLORS**. A client may elect to preserve these values when it installs its color map. Note that this mechanism is a way an for application to avoid the "technicolor" problem, whereby useful screen objects (e.g., window borders and title bars) vanish when the window manager installs a program's custom colors. For example:

```
SaveColor
{
        BorderColor
        TitleBackground
        TitleForeground
        "red"
        "green"
        "blue"
}
```

This places on the root window three pixel values for borders and title bars, as well as the three color strings, all taken from the default color map.

ShowIconManager
Display the icon manager's window when the window manager comes up. This window can always be brought up by invoking the function **f.showiconmgr**.

SortIconManager
Sort the entries in icon manager alphabetically, rather than by simply appending new windows to the end.

SqueezeTitle [{ *squeeze-list* **}]**
Use the **SHAPE** extension to confine title bars to the screen space they need, rather than letting them extend across the top of the window. The optional *squeeze-list* controls the location of the squeezed title bar along the top of the window. It contains entries of the form:

"*name*" *justification num denom*

where *name* names a window, *justification* is **left**, **center**, or **right**, and *num* and *denom* are numbers that specify a ratio that gives the relative position about which the title bar is justified. The ratio is measured from left to right if the numerator is positive, and right to left if negative. A denominator of zero indicates that the numerator should be measured in pixels. For convenience, the ratio 0/0 is the same as 1/2 for **center** and -1/1 for **right**. For example:

```
SqueezeTitle
{
      "XTerm"   left   0  0
      "xterm1"  left   1  3
      "xterm2"  left   2  3
      "oclock"  center 0  0
      "emacs"   right  0  0
}
```

You can use the variable **DontSqueezeTitle** to turn off squeezing for selected titles.

StartIconified [{ *win-list* **}]**
Leave client windows as icons until the user explicitly de-iconifies them. If the optional *win-list* is given, only the windows named therein begin life as icons. This option is useful for programs that do not support the command-line option **-iconic**.

TitleBackground *string* **[{** *win-list* **}]**
Set the background color for title bars. It can be specified only within a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **white**.

TitleButtonBorderWidth *pixels*
Give the width, in pixels, of a title button's border. This is typically set to zero, to allow title buttons to take up as much space as possible. The default is one.

TitleFont *string*
Name the font used when displaying window names in title bars. The default font is **variable**.

TitleForeground *string* **[{** *win-list* **}]**
Name the foreground color to use in title bars. It can be specified only within a **Color** or **Monochrome** list. The optional *wincolorlist* lets you set the color for specific windows, as described above for **BorderColor**. The default color is **black**.

TitlePadding *pixels*
Give the distance between the various buttons, text, and highlight areas in the title bar. The default is eight pixels.

UnknownIcon *string*
Name the bit-map file to be used as the default icon. This bit-map is used as the icon for every clients that does not provide an icon bit-map and is not named in the **Icons** list.

UsePPosition *string*
Honor program-requested locations (given by the flag **PPosition** in the property **WM_NORMAL_HINTS**) in the absence of a user-specified position. The argument *string* can be one of the following three values:

off Ignore the program-supplied position. This is the default.
on Use the position.
non-zero
Use the position if it is other than (0,0).

This option is for working around a bug in older toolkits.

WarpCursor [{ *win-list* **}]**
Warp the mouse cursor into a window when it is de-iconified. If the optional *win-list* is given, **twm** jumps the cursor only into the windows named therein.

WindowRing { *win-list* }
List the windows along which the function **f.warpring** cycles.

WarpUnmapped
Force function **f.warpto** to de-iconify a window when it jumps the mouse cursor into it. This typically is used to create a key binding that pops up a window no matter where it is. The default is for **f.warpto** to ignore iconified windows.

XorValue *number*
Set the value to use when drawing window outlines for moving and resizing. Set this to a value that results in a variety of distinguishable colors when exclusive-OR'ed with the contents of a typical user's screen. Setting this variable to one often gives nice results if adjacent colors in the default color map are distinct. By default, **twm** attempts to draw the temporary lines at the opposite end of the color map from the graphics.

Zoom [*count* **]**
"Zoom" a window when it is iconified or de-iconified. Whenever a window is iconified or de-iconified, **twm** rapidly displays a sequence of gradually enlarging outlines that suggest the window's movement to and from its iconified state. *count* gives the number of outlines to draw; the default is eight.

The following variables must be set after the fonts have been assigned; tberefore, it is best to put them at the end of the variables section:

DefaultFunction *function*
Execute *function* when **twm** receives a key or button event for which no binding is provided. This typically is bound to **f.nop**, to **f.beep**, or to a menu that contains window operations.

WindowFunction *function*
Execute *function* when the user selects a window from menu **TwmWindows**. If this variable is not set, **TwmWindows** is de-iconified and raised.

Bindings

After the variables have been set, your start-up file can have a section of *bindings*. These link functions to title buttons, mouse buttons, and keys on the keyboard.

Title buttons can be added from the left or right side, and appear in the title bar from left-to-right, according to the order in which they are specified. Key and mouse-button bindings can appear in any order.

A title-button's specifications must name the pixel map to use in the button box and the function to invoke when the user presses a mouse button within it. For example:

```
LeftTitleButton "bitmapname" = function
```

or

```
RightTitleButton "bitmapname" = function
```

bitmapname can name one of the built-in bit-maps (which are scaled to match **TitleFont**) by using the appropriate colon-prefixed name, described above.

Key and mouse-button specifications give the modifiers that must be pressed, the parts of the screen over which the mouse must be positioned, and the function to invoke when the event occurs. Keys are given as strings that contain the appropriate keysym name; buttons are given as the keywords **Button1** through **Button5**. For example:

```
"FP1"   = modlist : context : function
Button1 = modlist : context : function
```

modlist is any combination of the following modifier names:

```
shift          control        lock
meta           mod1           mod2
mod3           mod4           mod5
```

These can be abbreviated as follows:

```
s              c              l
m              m1             m2
m3             m4             m5
```

LEXICON

Each entry must be separated by a vertical bar '|'.

Likewise, *context* is any combination of the following variables:

```
window          title           icon
root            frame           iconmgr
```

Each of the above can be abbreviated. The entries must be separated by a vertical bar. The entry **all** combines all of the above variables.

function is any of the functions described below.

For example, the following gives the bindings from the default start-up file:

```
Button1     =   : root          : f.menu "TwmWindows"
Button1     = m : window | icon : f.function "move-or-lower"
Button2     = m : window | icon : f.iconify
Button3     = m : window | icon : f.function "move-or-raise"
Button1     =   : title         : f.function "move-or-raise"
Button2     =   : title         : f.raiselower
Button1     =   : icon          : f.function "move-or-iconify"
Button2     =   : icon          : f.iconify
Button1     =   : iconmgr       : f.iconify
Button2     =   : iconmgr       : f.iconify
```

A user who wanted to manipulate windows from the keyboard could use the following bindings:

```
"F1"    =       : all    : f.iconify
"F2"    =       : all    : f.raiselower
"F3"    =       : all    : f.warpring "next"
"F4"    =       : all    : f.warpto "xmh"
"F5"    =       : all    : f.warpto "emacs"
"F6"    =       : all    : f.colormap "next"
"F7"    =       : all    : f.colormap "default"
"Left"  = m     : all    : f.backiconmgr
"Right" = m | s : all    : f.forwiconmgr
"Up"    = m     : all    : f.upiconmgr
"Down"  = m | s : all    : f.downiconmgr
```

In the above example, the keys **Left**, **Right**, **Up**, and **Down** invoke, respectively, the keys (æ), (Æ), (ª), and (°). **twm** provides many more window-manipulation primitives than can be conveniently stored in a title bar, menu, or set of key bindings. Although a small set of defaults are supplied (unless the **NoDefaults** is specified), most users will want to bind their most common operations to key and button strokes. To do this, **twm** associates names with each of the primitives and provides *user-defined functions* for building higher-level primitives, and *menus* for interactively selecting among groups of functions.

A user-defined function consists of the name by which it is referenced in calls to **f.function**, and a list (enclosed in braces) of the functions to execute when this function is invoked. For example:

```
Function "move-or-lower"    { f.move f.deltastop f.lower }
Function "move-or-raise"    { f.move f.deltastop f.raise }
Function "move-or-iconify"  { f.move f.deltastop f.iconify }
Function "restore-colormap" { f.colormap "default" f.lower }
```

You must use the new function's name in **f.function** exactly as it appears in its specification.

In the following descriptions, if the function is said to operate on the selected window but is invoked from a root menu, the mouse cursor changes to the **Select** cursor and **twm** executes the function on the next window to receive a mouse-button click:

! *string* This is an abbreviation for **f.exec** *string*.

f.autoraise Toggle whether to raise the selected window when the mouse cursor enters it. See the description of the variable **AutoRaise**, above.

f.backiconmgr
Warp the mouse cursor into the previous column in the current icon manager. If necessary, wrap back to the previous row.

f.beep Sound the bell.

f.bottomzoom
This function resembles the function **f.fullzoom**, but resizes the window to fill only the bottom half of the screen.

f.circledown
Lower the top-most window that occludes another window.

f.circleup Raise the bottom-most window that is occluded by another window.

f.colormap *string*
Rotate the color maps (obtained from the property **WM_COLORMAP_WINDOWS** on the window) that **twm** displays when the mouse cursor is in this window. *string* can be one of the following values: **next**, **prev**, and **default**. Note that, in general, the installed color map is determined by the keyboard focus. A mouse-driven keyboard focus installs a private color map when it enters the window that owns that color map. Using the click-to-type model, **twm** does not install a private color map until the user presses a mouse button on the target window.

f.deiconify De-iconify the selected window. If the window is not an icon, this function does nothing.

f.delete Sends the message **WM_DELETE_WINDOW** to the selected window if the client application has requested it through the window property **WM_PROTOCOLS**. The application should respond to the message by removing the indicated window. If the window has not requested **WM_DELETE_WINDOW** messages, **twm** beeps to indicate that the user should choose an alternative method. Note this is very different from the behavior of the function **f.destroy**. The intention here is to delete one window, not necessarily the entire application.

f.deltastop Abort a user-defined function if the mouse cursor moves more than *MoveDelta* pixels. See the example definition given for function **move-or-raise** at the beginning of the section.

f.destroy Tell the X server to close the display connection of the client that created the selected window. This should only be used as a last resort for shutting down runaway clients. For a kinder, gentler approach to terminating applications, see the description of function **f.delete**, above.

f.downiconmgr
Warp the mouse cursor to the next row in the current icon manger. Wrap to the beginning of the next column, if necessary.

f.exec *string*
Pass *string* to **/bin/sh** to execute. In multiscreen mode, if *string* starts a new X client without giving a display argument, the client appears on the screen from which this function was invoked.

f.focus Toggle the keyboard focus of the server into the selected window; if necessary, change the focus rule from mouse-driven. If the selected window is already focused, execute function **f.unfocus**.

f.forcemove
This function resembles function **f.move**, except that it ignores the variable **DontMoveOff**.

f.forwiconmgr
Warp the mouse cursor to the next column in the current icon manager. Wrap to the beginning of the next row if necessary.

f.fullzoom Resize the selected window to the full size of the display. If the window is already zoomed, restore it to its original size.

f.function *string*
Executes the user-defined function *string*.

f.hbzoom This is a synonym for **f.bottomzoom**.

f.hideiconmgr
Unmap the current icon manager.

f.horizoom This function resembles function **f.zoom**, except that it resizes the selected window to the full width of the display.

f.htzoom This is a synonym for **f.topzoom**.

f.hzoom This is a synonym for **f.horizoom**.

f.iconify Iconify the selected window or icon. If the selected window or icon already is iconified, then de-iconify it.

f.identify Displays the selected window's name and geometry. Clicking the mouse or pressing a key in the window erases the displayed information.

f.lefticonmgr
This function resembles function **f.backiconmgr**, except that wrapping does not change rows.

f.leftzoom This function resembles function **f.bottomzoom**, except that it resizes the selected window only to the left half of the display.

f.lower Lower the selected window.

f.menu *string*
Invoke the menu *string*. You can build cascaded menus by nesting calls to **f.menu**.

f.move Drag an outline of the selected window (or the window itself, should variable **OpaqueMove** be set) until the user releases the invoking button on the mouse. Double-clicking within the number of milliseconds given by variable **ConstrainedMoveTime** jumps the mouse cursor to the center of the window and constrains the move to be either horizontal or vertical, depending upon which grid line is crossed. To abort a move, press another button before releasing the first button.

f.nexticonmgr
Warp the mouse cursor to the next icon manager that contains any windows on the current or any succeeding screen.

f.nop Do nothing. This function typically is used with the variables **DefaultFunction** and **WindowFunction**, and to introduce blank lines in menus.

f.previconmgr
Warp the mouse cursor to the previous icon manager that contains any windows on the current or preceding screens.

f.quit Restore the window's borders and exit from **twm**. If **twm** is the first client invoked from **xdm**, this function resets the server.

f.raise Raise the selected window.

f.raiselower
Raise the selected window to the top of the stacking order if it is occluded by any windows. If the window is not occluded, lower it.

f.refresh Refresh all windows.

f.resize Display an outline of the selected window. Crossing a border (or setting the variable **AutoRelativeResize**) stretches the outline like a rubber band until the user releases the invoking button on the mouse. To abort a resize, press another mouse button before you release the first.

f.restart Kill and restart **twm**.

f.righticonmgr
This function resembles the function **f.nexticonmgr**, except that wrapping does not change rows.

f.rightzoom
This function resembles the function **f.bottomzoom**, except that the selected window is resized only to the right half of the display.

f.saveyourself
Send the message **WM_SAVEYOURSELF** to the selected window if it has requested the message in its window property **WM_PROTOCOLS**. Clients that accept this message are expected to checkpoint all states associated with the window and update the property **WM_COMMAND**, as specified in the ICCCM. If the selected window has not selected for this message, beep.

f.showiconmgr
Map the current icon manager.

f.sorticonmgr
Sort the entries in the current icon manager alphabetically. See the description of the variable **SortIconManager**, above.

f.source *file*
Read and parse *file* as a **twm** startup file. You should use this function only to re-build your pull-down menus. None of the **twm** variables change.

f.title Provide a centered, unselectable item in a menu definition. Do not use it in any other context.

f.topzoom This function resembles function **f.bottomzoom**, except that it resizes the selected window only to the top half of the display.

f.twmrc Re-read the startup customization file. This function resembles function **f.source**, except that you are not required to name the file that **twm** is to read.

f.unfocus Reset the focus to mouse-driven. This should be used when a focused window is no longer desired.

f.upiconmgr
Warp the mouse cursor to the previous row in the current icon manager. If necessary, wrap to the last row in the same column.

f.version Display a window that shows the version of **twm** that you are using. The window remains displayed until you either press mouse button, you move the mouse cursor from one window to another.

f.vlzoom This is a synonym for function **f.leftzoom**.

f.vrzoom This is a synonym for function **f.rightzoom**.

f.warpring *string*
Jump (or "warp") the mouse cursor to the next or previous window. *string* indicates the direction in which to jump: either **next** or **prev**. See the description of the variable **WindowRing**, above.

f.warpto *string*
Jump (or "warp") the mouse cursor to the window whose name or class matches *string*. If the window is iconified, de-iconify it if the variable **WarpUnmapped**, **or ignore it if that variable is not set.**

f.warptoiconmgr *string*
Jump (or "warp") the mouse cursor from a window to that window's entry in icon manager *string*. If *string* is empty (i.e., ""), jump to the icon in the current icon manager.

f.warptoscreen *string*
Warp the mouse cursor to the screen *string*, which can be any of the following:

- A number (e.g., "0" or "1").
- The word **next**, which indicates the current screen plus one, skipping over any unmanaged screens.
- The word **back**, which indicates the current screen minus one, skipping over any unmanaged screens.
- The word **prev**, which indicates the last screen visited.

f.winrefresh
This function resembles the function **f.refresh**, except that it refreshes only the selected window.

f.zoom This function resembles the function **f.fullzoom** except that it changes only the height of the selected window.

Menus

You can group one or more functions into a menu. The user can select interactively one of the menu's entries; when an entry is selected, **twm** executes it.

Menus come in two flavors: *pop-up*, when the menu is bound to a mouse button; and *pull-down*, when it is bound to a title button.

A menu, as you define it, has the following syntax:

Menu "*menuname*" **[** ("*forecolor*":"*backcolor*") **]**
{
name1 **[** ("*fore*":"*back*")**]** *function1*
name2 **[** ("*fore*":"*back*")**]** *function2*
.
.
.
nameN **[** ("*foreN*":"*backN*")**]** *functionN*
}

The keyword **Menu** tells **twm** that the following is a menu. This is followed by the name of the menu. You can pass this name to the function **f.menu** (described above) to invoke this menu. Note that you must reproduce the menu name *exactly*, matching both the case of every character and white space within the name (if any). The name is followed by optional default foreground and background colors for each entry in the menu.

Then comes the body of the menu, which is enclosed between braces. The body of the menu consists of an indefinite number of entries, one for each row in the menu. The field *name* give the name of the entry, as it appears in the menu. The optional fields *fore* and *back* give the foreground and background colors to use with this entry. These colors are used only on a color display. The default is to use the colors specified by the variables **MenuForeground** and **MenuBackground**. Finally, the field *function* names the function (or functions) that **twm** is to execute when the user selects that menu entry. This can include any user-defined functions or additional menus.

The following gives an example menu, which invokes some X clients:

```
Menu "Applications" ("black":"lightseagreen")
{
        "APPLICATIONS" ("black":"lightseagreen") f.title
        "Font Select"  ! "xfontsel &"
        "XBiff"        ! "xbiff &"
        "XCalc"        f.menu " Calculator "
        "XClock"       ! "xclock -chime -fg blue -update 1&"
        "Xeyes"        ! "xeyes -geometry 200x200+150+150 -fg red&"
        "XLoad"        ! "xload &"
        "XLogo"        ! "xlogo &"
        "XTerm"        ! "xterm  -T 'BOURNE SHELL' -n 'BOURNE SHELL' -e sh &"
        "XTetris"      ! "xtetris &"
}
```

The menu is named **Applications**. The entries in the menu are given the default colors of green characters on a black background.

The first entry in the menu calls the function **f.title**, which displays the menu's title. This entry sets the title's colors to black characters on a green background, so it will stand out.

The entry **XCalc** calls the function **f.menu** to display another menu, in this case the one named **Calculator**. Note that the quotations marks around the menu's name includes white space; this is because the author of the menu **Calculator** chose (for some reason) to embed white space in its name, and those spaces must be reproduced here.

The other entries invoke the function '!', which (as noted above) is a synonym for the COHERENT command **/bin/sh**. The text that follows names the command you want **sh** to execute (in each case, an X client), plus its arguments. Each commands ends in an '&', which tells **sh** to execute the client in the background. (If the command were executed in the foreground, you would be stuck in this menu until you managed to kill the client that command invoked.) Note that most X clients live in directory **/usr/X11/bin**; if the user who invoked **twm** does not name this directory in her **PATH**, **sh** will not find the client and the command will fail.

A special menu named **TwmWindows** names all of the windows supplied by **twm** and its clients. Selecting an entry tells **twm** to execute on that window the function defined by the variable **WindowFunction**. If **WindowFunction** has not been set, **twm** de-iconifies and raises the window.

Icons

twm supports several ways to manipulate iconified windows. The common pixel map-and-text style may be laid out by hand or automatically arranged, as described by the variable **IconRegion**. In addition, a terse grid of icon names, called an *icon manager*, uses screen space more efficiently and lets the user navigate amongst windows from the keyboard.

An icon manager is a window that names windows. Each entry in the icon manager is a small rectangle that contains the window's name. When a window is iconified, the icon manager also displays the **iconify** symbol to the left of the window's name. By default, every client that you invoke is given an entry in the icon manager. You can, however, exclude selected windows from the icon manager by naming them in the variable **IconManagerDontShow** (described above).

When you move the mouse cursor into an entry in the icon manager, the icon manager also directs keyboard focus to the corresponding window. By invoking the functions **f.upiconmgr**, **f.downiconmgr**, **f.lefticonmgr**, and **f.righticonmgr** from the keyboard, you can move the input focus directly from the keyboard.

By default, clicking on an entry in the icon manager invokes the function **f.iconify** (described above). To change the actions taken in the icon manager, use the the context **iconmgr** when you specify button and keyboard bindings.

Environment Variables

twm reads the following environmental variables:

DISPLAY
Name the X server to use. It is also set during **f.exec**, so programs appear on the proper screen.

HOME Give the prefix for files that begin with a tilde, and for locating the **twm** start-up file.

Files

$HOME/.twmrc — Personalized configuration file
/usr/X11/lib/twm/system.twmrc — Default configuration file

See Also

X utilities, xdm, xinit, xrdb

Notes

Double clicking very quickly to obtain the constrained move function sometimes causes the window to move, even though the mouse cursor is not moved.

If **IconifyByUnmapping** is on and windows are listed in **IconManagerDontShow** but not in **DontIconifyByUnmapping**, they may be lost if they are iconified and no bindings are set to **f.menu** or **f.warpto**.

twm was written by Tom LaStrange of Solbourne Computer; Jim Fulton, Keith Packard, and Dave Sternlicht of the MIT X Consortium; Steve Pitschke of Stardent Computer; and Dave Payne of Apple Computer.

viewres — X Utility

Graphical class browser for Xt
viewres [-*option ...***]**

viewres displays a tree that show the hierarchy of widget classes of the Athena Widget Set. It can expand each node in the tree to show the resources that the corresponding class adds (i.e., does not inherit from its parent) when a widget is created.

viewres recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-top *name* Display widget *name* as the highest widget in the hierarchy. You can use this option to limit the display to a subset of the tree. The default is **Object**.

-variable Display in each node the widget's variable name (as declared in its header file) instead of its class name. This can help you distinguish widget classes that have the same name (e.g., **Text**).

-vertical Display the tree from top to bottom on your screen, instead of from left to right (the default).

-xrm *resourcestring*
Use *resourcestring* to define a resource.

View Menu

You can invoke the following commands from **viewres**'s menu **View** to change the way it displays the widget tree:

`(Show_Variable_Names)`
Set the label on each node to the variable name used to declare the corresponding widget class. You can also perform this operation via the translation **SetLabelType(**variable**)**.

`(Show_Class_Names)`
Set the label on each node to the class name used when specifying a resource. You can also perform this operation via the translation **SetLabelType(**class**)**.

`(Layout_Horizontal)`
Display the tree from left to right on your screen. You can also perform this operation via the translation **SetOrientation(West)**.

`(Layout_Vertical)`
Display the tree from top to bottom on your screen. You can also perform this operation via the translation **SetOrientation(North)**.

`(Show_Resource_Boxes)`
Expand the selected nodes (see next section) to show the new widget and constraint resources. You can also perform this operation via the translation **Resources(on)**.

`(Hide_Resource_Boxes)`
Remove the resource displays from the selected nodes (usually to conserve space). You can also perform this operation via the translation **Resources(off)**.

Select Menu

To display the resources for one widget class, either move the mouse cursor to the corresponding node and click the middle mouse button, or use the right mouse button to add the node to the selection list and then invoke entry `(Show_Resource_Boxes)` from menu **View**. Because the left mouse button toggles the selection state of a node, clicking on a selected node removes it from the selected list.

You can also use the following commands on the menu **Select** to collect nodes:

`(Unselect_All)`
Remove all nodes from the selection list. You can also perform this operation via the translation **Select(nothing)**.

`(Select_All)`
Add all nodes to the selection list. You can also perform this operation via the translation **Select(all)**.

`(Invert_All)`
Invert the select status of all nodes: nodes that are unselected become selected, and selected nodes that are selected become unselected. You can also perform this operation via the translation **Select(invert)**.

`(Select_Parent)`
Select the immediate parents of all selected nodes. You can also perform this operation via the translation **Select(parent)**.

(Select_Ancestors)
Recursively select all parents of all selected nodes. You can also perform this operation via the translation **Select(ancestors)**.

(Select_Children)
Select the immediate children of all selected nodes. You can also perform this operation via the translation **Select(children)**.

(Select_Descendants)
Recursively select all children of all selected nodes. You can also perform this operation via the translation **Select(descendants)**.

(Select_Has_Resources)
Select all nodes that add new resources (regular or constraint) to their corresponding widget classes. You can also perform this operation via the translation **Select(resources)**.

(Select_Shown_Resource_Boxes)
Select all nodes whose resource boxes are currently expanded (usually so that they can be closed by **Hide Resource Boxes**). You can also perform this operation via the translation **Select(shown)**.

Actions

viewres lets you perform the following actions:

Quit() Exit from **viewres**.

SetLabelType(*type***)**
Set what the node labels display. *type* can be either **class** or **variable**.

SetOrientation(*direction***)**
Plant the root of the tree onto the *direction* edge of your screen. **viewres** then grows the tree toward the opposite edge of the screen. *direction* can be one of the following: **West**, **North**, **East**, or **South**.

Select(*what***)**
Select the indicated nodes. *what* indicates the nodes to select, as follows: **nothing**, **invert**, **parent**, **ancestors, children**, **descendants**, **resources**, or **shown**. Each is describe in the section on the **View** menu, above.

Resources(*op***)**
Change the state of the resource boxes in the selected nodes. *op* can be **on**, **off**, or **toggle**. If invoked from within one of the nodes (through the keyboard or pointer), only that node is affected.

Widget Hierarchy

Resources may be specified for the following widgets. *variable-name* is the widget variable name of each node:

```
Viewres viewres
      Paned pane
            Box buttonbox
                  Command quit

                  MenuButton view
                        SimpleMenu viewMenu
                              SmeBSB layoutHorizontal
                              SmeBSB layoutVertical
                              SmeLine line1
                              SmeBSB namesVariable
                              SmeBSB namesClass
                              SmeLine line2
                              SmeBSB viewResources
                              SmeBSB viewNoResources
```

```
            MenuButton select
                  SimpleMenu selectMenu
                        SmeBSB unselect
                        SmeBSB selectAll
                        SmeBSB selectInvert
                        SmeLine line1
                        SmeBSB selectParent
                        SmeBSB selectAncestors
                        SmeBSB selectChildren
                        SmeBSB selectDescendants
                        SmeLine line2
                        SmeBSB selectHasResources
                        SmeBSB selectShownResources

      Form treeform
            Porthole porthole
                  Tree tree
                        Box variable-name
                              Toggle variable-name
                              List variable-name
            Panner panner
```

See Also

appres, editres, listres, X utilities

Notes

viewres was written by Jim Fulton of the MIT X Consortium.

X — X Utility

X Window System server
X [:*displaynumber***] [-***option ...***] [***ttyname***]**

X is the generic name for the X Window System server. It usually is a link to the appropriate server binary for driving the screen on your machine: **/usr/X11/bin/X386color** for VGA color, or **/usr/X11/bin/X386mono** for monochrome graphics systems.

Starting the Server

To launch the server, invoke the script **/usr/X11/bin/startx**. This script in turn invokes the X utility **xinit** with the appropriate configuration files. When the X server starts up, it takes over the console. You cannot log into the console while the server is running, although you can use the X client **xterm** to create a terminal within a window on your screen, and so give commands to your machine.

Network Connections

The X server by design supports connections via TCP/IP, DECnet, and the local UNIX domain. However, the COHERENT implementation of X does not yet support networking. The server at this point can interact only with one display: your machine's console.

Options

All X servers accept the following command-line options. Note that the ones that apply to networking do not yet apply to the COHERENT implementation of X, and therefore you should ignore them. You can use the X utility **xset** to change many of these parameters while **X** is in operation:

-a *number* Set pointer acceleration, i.e., the ratio of how much the mouse cursor moves to how much the user actually moved the mouse.

-ac Disable host-based access-control mechanisms. This enables access by any host, and permits any host to modify the access-control list. Use this option with extreme caution. It exists primarily for running test suites remotely.

-auth *file* *file* contains a collection of authorization records used to authenticate access.

bc Disable certain kinds of error checking. This re-enables certain bugs that have been fixed in this release of the server, to support applications that need those bugs to run correctly.

-bs Disable backing storage on all screens.

-c Turn off key-click.

c *volume* Set the volume of the key-click. *volume* is a percent of the maximum volume.

-cc *class* Set to *class* the visual class for the root window of color screens. The class numbers are as specified in the X protocol. Not every server obeys this option.

-co *filename*
Read the RGB color data base from *filename*. The default is **/usr/X11/lib/rgb.txt**.

-dpi *resolution*
Set the resolution of the screen, in dots per inch. Use this option when the server cannot derive the screen size from the hardware.

-f *volume* Set the volume of the bell. This is a percent of the maximum volume.

-fc *cursorfont*
Set the font for the mouse cursor. The default is **cursor**.

-fn *font* Use *font* as the default text font.

-fp *fontPath* Set the search path for fonts. This path is a comma-separated list of directories that the X server searches for font data bases.

-help Print a usage message.

-I Ignore all remaining command-line arguments.

-logo Turn on display of the X Window System's logo in the screen saver. This cannot be changed from within a client.

nologo Do not display the X Window System's logo in the screen saver. This cannot be changed from within a client. This is the default.

-p *minutes* Set to *minutes* the screen-saver's time of cycling its pattern.

-probeonly Interrogate the video card to discover its chip set and amount of video RAM, but do not bring up X.

-r Turn off auto-repeat.

r Turn on auto-repeat.

-s *minutes* Invoke the screen-saver *minutes* after receiving the last input from the keyboard or mouse.

-su Disable save-under support on all screens.

-t *number* Set the threshhold of mouse-cursor (pointer) acceleration, in pixels: that is, let acceleration take effect after the user has moved the mouse cursor *number* pixels.

-to *seconds* Set the default connection's timeout *seconds*.

v When the screen saver is invoked, turn off video.

-v When the screen saver is invoked, leave video on.

-wm Force the default backing-store of all windows to be **WhenMapped**. This is one way to get backing-store to apply to all windows.

-x *extension*
Load *extension* at the time of initialization.

Security

The X server implements a simplistic authorization protocol, **MIT-MAGIC-COOKIE-1**, which uses data private to authorized clients and the server. This is a trivial scheme: if the client passes authorization data that are the same as those possessed by the server, it is allowed access. This scheme is worse than the host-based access control mechanisms in environments with unsecure networks as it allows any host to connect, given that it has discovered the private key; but in many environments, this level of security is better than the host-based scheme as it allows access control per user instead of per host.

In addition, the server provides support for a DES-based authorization scheme, **XDM-AUTHORIZATION-1**, which

is more secure (given a secure key distribution mechanism). This authorization scheme can be used in conjunction with XDMCP's authentication scheme (**XDM-AUTHENTICATION-1**) or in isolation.

The authorization data are passed to the server in a private file named by the command-line option **-auth**. Each time the server is about to accept the first connection after a reset (or when the server is starting), it reads this file. If this file contains any authorization records, the local host is not automatically allowed access to the server, and only clients which send one of the authorization records contained in the file in the connection setup information will be allowed access.

The X protocol intrinsically does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen. Sites that have better authentication and authorization systems (such as Kerberos) might wish to make use of the hooks in the libraries and the server to provide additional security models.

Signals

The X server attaches special meaning to the following signals:

SIGHUP
: Close all existing connections, free all resources, and restore all defaults. It is sent by the display manager whenever the main user's main client (usually **xterm** or the window manager **twm**) exits. This forces the server to clean up and prepare for its next invocation.

SIGTERM
: Exit cleanly.

SIGUSR1
: This signal is used quite differently from either of the above. When the server starts, it checks to see if it has inherited **SIGUSR1** as **SIG_IGN** instead of the usual **SIG_DFL**. In this case, the server sends **SIGUSR1** to its parent process after it has set up the various connection schemes.

Fonts

Fonts usually are stored as individual files in directories. The X server can obtain fonts from directories or from font servers. The list of directories and font servers the X server uses when trying to open a font is controlled by the **FontPath**, which is set in the file **/usr/X11/lib/Xconfig**. Although most sites start up the X server with the appropriate font path (using the option **-fp**, mentioned above), it can be overridden using the X utility **xset**.

The default font path for the X server contains the following directories:

/usr/X11/lib/fonts/misc
: This directory contains miscellaneous bit-mapped fonts. It also has various font-name aliases for the fonts.

/usr/X11/lib/fonts/75dpi
: This directory contains bit-mapped fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow & Holmes, and Sun Microsystems, Inc., for 75 dot-per-inch displays. It contains integrated selection of sizes, styles, and weights for each family of typefaces.

The program **mkfontdir** create a font data base in the directory that contains the compiled versions of the fonts (the **.pcf** files). Whenever you add a font to a directory, run **mkfontdir** so that the server can find the new font.

Files

/usr/X11/bin/X — Default X server
/usr/X11/bin/X386color — Color X server
/usr/X11/bin/X386mono — Monochrome X server
/usr/X11/lib/Xconfig — Configuration file for X server
/usr/X11/lib/fonts/75dpi — Directory of bit-mapped fonts
/usr/X11/lib/fonts/misc — Directory of miscellaneous fonts
/usr/X11/lib/rgb.txt — Color data base

See Also

fs, mkfontdir, startx, twm, xauth, xinit, xset, xsetroot, xterm

Notes

The sample server was originally written by Susan Angebranndt, Raymond Drewry, Philip Karlton, and Todd Newman of Digital Equipment Corporation, with support from many other persons. It has since been extensively rewritten by Keith Packard and Bob Scheifler of MIT.

X clients — Overview

In the contents of the X Window System, a *client* is a program that runs under the tutelage of the X Windows server. In this manual, a client means a program that *does something* under the X Windows server other than help run the system.

Games

The following clients are just for fun:

maze . . . . . Create and solve a random maze
puzzle . . . . . The X scrambled-number game
xgas . . . . . Animated simulation of an ideal gas
xtetris . . . . . Wildly amusing implementation of Tetris

Monitoring the System

These clients help you observe the operation of your system:

xbiff . . . . . Notify the user that mail has arrived
xeyes . . . . . Display two roving eyes
xload . . . . . Display your system's load average

Pretty Pictures

The main attraction of X is, of course, its graphical capabilities. The following clients help you to manipulate pretty pictures:

ico . . . . . Animate an icosahedron or other polyhedron
xlogo . . . . . Display the X Window System logo
xmag . . . . . Magnify a part of the screen
xgc . . . . . X graphics demonstration

Timepieces

X Windows for COHERENT includes two clients that display the current time:

oclock . . . . . Display an analogue clock
xclock . . . . . Display a clock

Tools

Finally, the following tools help you do useful work under X:

xcalc . . . . . Scientific calculator for X
xedit . . . . . Simple text editor for X
xpr . . . . . Print a dump of an X window
xterm . . . . . Terminal emulator for X
xvt . . . . . VT100 emulator
xwd . . . . . Dump an image of an X window
xwud . . . . . Un-dump a window image

See Also

X utilities

X utilities — Overview

In this manual, an X *utility* is a program that helps you to manage the X Window System.

Bit Maps

The following utilities help you to manage bit-mapped images:

atobm . . . . . Convert ASCII to an X bit-mapped image
bitmap . . . . . Bit map editor
bmtoa . . . . . Convert an X bit-mapped image to ASCII

Colors

The following utilities help you manage your system's color palette:

showrgb . . . . . Un-compile an RGB color-name data base
xcmsdb. . . . . Manipulate **xlib** screen-color characterization data
xstdcmap . . . . . X standard color-map utility

Fonts

The following utilities display and list fonts:

bdftopcf . . . . . Generate a PCF font from a BDF file
mkfontdir . . . . . Create file **fonts.dir** from directory of font files
xfd . . . . . Display all the characters in an X font
xfontsel . . . . . Interactively select X11 fonts
xlsfonts . . . . . List fonts being used on a server

Modifying the Screen

These utilities let you modify your screen "on the fly":

xrefresh . . . . . Refresh all or part of an X screen
xset . . . . . Set preferences for the display
xsetroot . . . . . Set preferences for the root window

Programming Tools

These utilities help you program under X. Note that these tools, in particular **makedepend**, can be used with non-X applications as well:

imake. . . . . C preprocessor interface to the **make** utility
makedepend . . . Create dependencies in **makefile**s
mkdirhier . . . . . Make a directory hierarchy
xmkmf . . . . . Create a **makefile** from an **Imakefile**

Resources

These utilities set and help you to manage resources:

appres . . . . . List an application's resource data base
editres . . . . . Resource editor for X Toolkit applications
listres. . . . . List resources in widgets
viewres. . . . . Graphical class browser for **Xt**
xprop . . . . . Display the X server's properties
xrdb. . . . . Read/set the X server's resource data base

System Monitoring

These utilities help you to monitor the operation of your system:

xauth . . . . . Display/edit authorization information
xdpyinfo . . . . . Display information about an X server
xev . . . . . Print contents of X events
xlsatoms . . . . . List interned atoms defined on server
xlsclients . . . . . List client applications running on a display
xwininfo . . . . . Display information about a window

Miscellaneous

Finally, the following utilities do not fit neatly into any other category. These include some of the most interesting (and important) tools in the X package:

resize . . . . . Set environmental variables to show window size
startx . . . . . Initiate an X session
twm . . . . . Tab Window Manager for the X Window System
xclipboard . . . . . Hold multiple selections for later retrieval
xcmstest . . . . . XCMS test program
xcutsel . . . . . Copy text between the cut buffer and the primary selection
xinit . . . . . Initiate the X Window System
xkill . . . . . Kill an X client
xmodmap . . . . . Modify X keymaps

See Also

X clients

x11perf — X Utility

Test performance of the X11 server
x11perf [-*option* **...]**

x11perf runs one or more performance tests and reports how quickly your X server can execute them

Many graphics benchmarks assume that the graphics device is used to display the output of a single fancy graphics application, and that the user gets his work done on some other device, like a terminal. Such benchmarks usually measure drawing speed for lines, polygons, text, etc. Because workstations are not used as stand-alone graphics engines, but as super-terminals, **x11perf** measures window-management performance as well as traditional graphics performance. It includes benchmarks for the time your server takes to create and map windows (as when you start up an application); to map a pre-existing set of windows onto the screen (as when you deiconify an application or pop-up a menu); and to rearrange windows (as when you slosh windows to and fro trying to find the one you want).

x11perf also measures graphics performance for operations not normally used in stand-alone graphics displays, but are nonetheless used frequently by X applications. Such operations include **CopyPlane** (used to map bitmaps into pixels), scrolling (used in text windows), and various stipples and tiles (used for CAD and color half-toning, respectively).

You should use **x11perf** to analyze the strengths and weaknesses of servers. It is most useful to the writer of a server who wants to analyze and improve it.

x11perf exercises nearly every X11 operation you can perform; it does not limit itself to a representative sample of the operations that X11 applications actually use. Although it can be used as a benchmark, it was written and is intended test performance. As such, **x11perf** *does not* whittle down measurements to a single "HeXStones" or "MeXops" number. We consider such numbers to be uninformative at best, and misleading at worst. Some servers that are very fast for certain applications can be very slow for others. No single number or small set of numbers are sufficient to characterize how an X implementation performs on all applications. However, by knowledge of your favorite application, you may be able to use the numbers **x11perf** reports to predict its performance on a given X implementation.

That being said, you may also want to look at **x11perfcomp** a program that compares the outputs of different runs of **x11perf**. You provide it a list of files that contain the results from **x11perf**, and it lays them out in a nice, tabular format.

For repeatable results, run **x11perf** using a local connection on a freshly started server. The default configuration runs each test five times, to see if each trial takes approximately the same amount of time. Strange glitches should be examined; if non-repeatable, one might chalk them up to daemons and network traffic. Each trial is run for five seconds, to reduce random differences in time. The number of objects processed per second is displayed to three significant digits, but you will be lucky on most UNIX systems if the numbers are consistent to two digits.

x11perf moves the cursor out of the test window. Be careful not to bump the mouse and move it back into the window. (A prize to the person who explains why!)

Before running a test, **x11perf** determines what the round trip time to the server is, and factors this out of the final timing reported. It ensures that the server has actually performed the work requested by fetching a pixel back from the test window; this ensures that servers that talk to graphics accelerators cannot claim that they have finished while the accelerator is still painting madly.

By default, **x11perf** automatically calibrates the number of repetitions of each test, so each should take

approximately the same length of time to run across servers of widely differing speeds. However, as each test must be run to completion at least once, some slow servers may take a very long time, particularly on the window moving and resizing tests, and on the arc drawing tests.

All timing reports are for the smallest object involved. For example, the line tests use a **PolyLine** request to paint several lines at once, but report how many lines per second the server can paint, not how many **PolyLine** requests per second. Text tests paint a line of characters, but report on the number of characters per second. Some window tests map, unmap, or move a single parent window, but report on how many children windows per second the server can map, unmap, or move.

Options

x11perf is based solely on **Xlib**. It recognizes the following command-line options:

-display *host*:*display*
Use *display* on *host*.

-sync Run the tests in synchronous mode. Normally, this is useful only for debugging **x11perf**

-pack Run rectangle tests so that they pack rectangles right next to each other. This makes it easy to debug server code for stipples and tiles: if the pattern looks ugly, you've got alignment problems.

-repeat *n* Repeats each test *n* times. By default, **x11perf** runs each test five times.

-time *seconds* Run each test for *seconds*. The default is five seconds for each test.

-all Runs every test. This may take a long time to execute.

-range *test1***[,***testN***]**
Run every test from *test1* through *testN*. The test names should be one of the options starting from **-dot**. These are described below. For example, the option

```
-range line100,dline10
```

executes every test from **line100** through **dline10**. The option

```
-range line100
```

begins with the 100-pixel line test and runs to the last test in the suite.

-labels Generate just the descriptive labels for each test specified. For details, see the Lexicon entry for **x11perfcomp**.

-fg *color* Use *color* to paint the foreground.

-bg *color* Use *color* to paint the background.

-rop *rop0 ... ropN*
Use the raster operations *rop0* through *ropN*. The default is **GXcopy**. This option only affects graphics benchmarks that actually use the graphics function.

-pm *pm0 ... pmN*
Use plane masks *pm0* through *pmN*. The default is **~0**. This option only affects graphics benchmarks that actually use the plane mask.

-depth *depth* Use a visual with *depth* planes per pixel. The default is the default visual.

The following options name the tests that **x11perf** can perform. They appear here in the order in which **x11perf** executes them. If you name the tests with the **-range** option, you must keep this order in mind. You can also ask **x11perf** to perform one or more tests individually, by naming each test on on the command line:

-dot Draw dots.
-rect1 Draw 1×1 solid-filled rectangle.
-rect10 Draw 10×10 solid-filled rectangle.
-rect100 Draw 100×100 solid-filled rectangle.
-rect500 Draw 500×500 solid-filled rectangle.
-srect1 Draw 1×1 transparent stippled rectangle, 8×8 stipple pattern.
-srect10 Draw 10×10 transparent stippled rectangle, 8×8 stipple pattern.
-srect100 Draw 100×100 transparent stippled rectangle, 8×8 stipple pattern.

-srect500 Draw 500×500 transparent stippled rectangle, 8×8 stipple pattern.
-osrect1 Draw 1×1 opaque stippled rectangle, 8×8 stipple pattern.
-osrect10 Draw 10×10 opaque stippled rectangle, 8×8 stipple pattern.
-osrect100 Draw 100×100 opaque stippled rectangle, 8×8 stipple pattern.
-osrect500 Draw 500×500 opaque stippled rectangle, 8×8 stipple pattern.
-tilerect1 Draw 1×1 tiled rectangle, 4×4 tile pattern.
-tilerect10 Draw 10×10 tiled rectangle, 4×4 tile pattern.
-tilerect100 Draw 100×100 tiled rectangle, 4×4 tile pattern.
-tilerect500 Draw 500×500 tiled rectangle, 4×4 tile pattern.
-bigsrect1 Draw 1×1 stippled rectangle, 161×145 stipple pattern.
-bigsrect10 Draw 10×10 stippled rectangle, 161×145 stipple pattern.
-bigsrect100 Draw 100×100 stippled rectangle, 161×145 stipple pattern.
-bigsrect500 Draw 500×500 stippled rectangle, 161×145 stipple pattern.
-bigosrect1 Draw 1×1 opaque stippled rectangle, 161×145 stipple pattern.
-bigosrect10 Draw 10×10 opaque stippled rectangle, 161×145 stipple pattern.
-bigosrect100 Draw 100×100 opaque stippled rectangle, 161×145 stipple pattern.
-bigosrect500 Draw 500×500 opaque stippled rectangle, 161×145 stipple pattern.
-bigtilerect1 Draw 1×1 tiled rectangle, 161×145 tile pattern.
-bigtilerect10 Draw 10×10 tiled rectangle, 161×145 tile pattern.
-bigtilerect100 Draw 100×100 tiled rectangle, 161×145 tile pattern.
-bigtilerect500 Draw 500×500 tiled rectangle, 161×145 tile pattern.
-eschertilerect1
Draw 1×1 tiled rectangle, 215×208 tile pattern.
-eschertilerect10
Draw 10×10 tiled rectangle, 215×208 tile pattern.
-eschertilerect100
Draw 100×100 tiled rectangle, 215×208 tile pattern.
-eschertilerect500
Draw 500×500 tiled rectangle, 215×208 tile pattern.
-seg1 Draw one-pixel, thin-line segment.
-seg10 Draw ten-pixel, thin-line segment.
-seg100 Draw 100-pixel, thin-line segment.
-seg500 Draw 500-pixel, thin-line segment.
-seg100c1 Draw 100-pixel, thin-line segment (one obscuring rectangle).
-seg100c2 Draw 100-pixel, thin-line segment (two obscuring rectangles).
-seg100c3 Draw 100-pixel, thin-line segment (three obscuring rectangles).
-dseg10 Draw ten-pixel, thin, dashed segment (three on, two off).
-dseg100 Draw 100-pixel, thin, dashed segment (three on, two off).
-ddseg100 Draw100-pixel, thin, double-dashed segment (three foreground, two background).
-hseg10 Draw ten-pixel, thin, horizontal-line segment.
-hseg100 Draw 100-pixel, thin, horizontal-line segment.
-hseg500 Draw 500-pixel, thin, horizontal-line segment.
-vseg10 Draw ten-pixel, thin, vertical-line segment.
-vseg100 Draw 100-pixel, thin, vertical-line segment.
-vseg500 Draw 500-pixel, thin, vertical-line segment.
-whseg10 Draw ten-pixel, wide, horizontal-line segment.
-whseg100 Draw 100-pixel, wide, horizontal-line segment.
-whseg500 Draw 500-pixel, wide, horizontal-line segment.
-wvseg10 Draw ten-pixel, wide, vertical-line segment.
-wvseg100 Draw 100-pixel, wide, vertical-line segment.
-wvseg500 Draw 500-pixel, wide, vertical-line segment.
-line1 Draw one-pixel, thin (width zero) line.
-line10 Draw ten-pixel thin line.
-line100 Draw 100-pixel thin line.
-line500 Draw 500-pixel thin line.
-dline10 Draw ten-pixel, thin, dashed line (three on, two off).
-dline100 Draw 100-pixel, thin, dashed line (three on, two off).
-ddline100 Draw 100-pixel, thin, double-dashed line (three foreground, two background).
-wline10 Draw ten-pixel line, line width one.
-wline100 Draw 100-pixel line, line width ten.

-wline500 Draw 500-pixel line, line width 50.
-wdline100 Draw 100-pixel dashed line, line width ten (30 on, 20 off).
-wddline100 Draw 100-pixel double-dashed line, line width ten (30 fg, 20 bg).
-orect10 Draw 10×10, thin rectangle outline.
-orect100 Draw 100-pixel, thin, vertical-line segment.
-orect500 Draw 500-pixel, thin, vertical-line segment.
-worect10 Draw 10×10, wide rectangle outline.
-worect100 Draw 100-pixel, wide, vertical-line segment.
-worect500 Draw 500-pixel, wide vertical line segment.
-circle1 Draw one-pixel-diameter, thin (line width zero) circle.
-circle10 Draw ten-pixel-diameter, thin circle.
-circle100 Draw 100-pixel-diameter, thin circle.
-circle500 Draw 500-pixel-diameter, thin circle.
-dcircle100 Draw 100-pixel-diameter, thin, dashed circle (three on, two off).
-ddcircle100 Draw 100-pixel-diameter, thin, double-dashed circle (three foreground, two background).
-wcircle10 Draw ten-pixel-diameter circle, line width one.
-wcircle100 Draw 100-pixel-diameter circle, line width ten.
-wcircle500 Draw 500-pixel-diameter circle, line width 50.
-wdcircle100 Draw 100-pixel-diameter dashed circle, line width ten (30 on, 20 off).
-wddcircle100 Draw 100-pixel-diameter, double-dashed circle, line width ten (30 foreground, 20 background).
-pcircle10 Draw ten-pixel-diameter, thin, partial circle, orientation and arc angle evenly distributed.
-pcircle100 Draw 100-pixel-diameter, thin, partial circle.
-wpcircle10 Draw ten-pixel-diameter, wide, partial circle.
-wpcircle100 Draw 100-pixel-diameter, wide, partial circle.
-fcircle1 Draw one-pixel-diameter, filled circle.
-fcircle10 Draw ten-pixel-diameter, filled circle.
-fcircle100 Draw 100-pixel-diameter, filled circle.
-fcircle500 Draw 500-pixel-diameter, filled circle.
-fcpcircle10 Draw ten-pixel-diameter, partial filled circle, chord fill, orientation and arc angle evenly distributed.
-fcpcircle100 Draw 100-pixel-diameter, partially filled circle, chord fill.
-fspcircle10 Draw ten-pixel-diameter, partially filled circle, pie-slice fill, orientation and arc angle evenly distributed.
-fspcircle100 Draw 100-pixel-diameter, partially filled circle, pie slice fill.
-ellipse10 Draw ten-pixel-diameter, thin (line width zero) ellipse, major and minor axis sizes evenly distributed.
-ellipse100 Draw 100-pixel-diameter, thin ellipse.
-ellipse500 Draw 500-pixel-diameter, thin ellipse.
-dellipse100 Draw 100-pixel-diameter, thin, dashed ellipse (three on, two off).
-ddellipse100 Draw 100-pixel-diameter, thin, double-dashed ellipse (three foreground, two background).
-wellipse10 Draw ten-pixel-diameter ellipse, line width one.
-wellipse100 Draw 100-pixel-diameter ellipse, line width ten.
-wellipse500 Draw 500-pixel-diameter ellipse, line width 50.
-wdellipse100 Draw 100-pixel-diameter dashed ellipse, line width ten (30 on, 20 off).
-wddellipse100 Draw 100-pixel-diameter, double-dashed ellipse, line width ten (30 foreground, 20 background).
-pellipse10 Draw ten-pixel-diameter, thin, partial ellipse.
-pellipse100 Draw 100-pixel-diameter, thin, partial ellipse.
-wpellipse10 Draw ten-pixel-diameter, wide, partial ellipse.
-wpellipse100 Draw 100-pixel-diameter, wide, partial ellipse.
-fellipse10 Draw ten-pixel-diameter filled ellipse.
-fellipse100 Draw 100-pixel diameter filled ellipse.
-fellipse500 Draw 500-pixel-diameter filled ellipse.
-fcpellipse10 Draw ten-pixel-diameter, partially filled ellipse, chord fill.
-fcpellipse100 Draw 100-pixel-diameter, partially filled ellipse, chord fill.
-fspellipse10 Draw ten-pixel-diameter, partially filled ellipse, pie-slice fill.
-fspellipse100 Draw 100-pixel-diameter, partially filled ellipse, pie-slice fill.
-triangle1 Fill one-pixel/side triangle.
-triangle10 Fill ten-pixel/side triangle.
-triangle100 Fill 100-pixel/side triangle.
-trap10 Fill 10×10 trapezoid.

-trap100 Fill 100×100 trapezoid.
-strap10 Fill 10×10 transparent stippled trapezoid, 8×8 stipple pattern.
-strap100 Fill 100×100 transparent stippled trapezoid, 8×8 stipple pattern.
-ostrap10 Fill 10×10 opaque stippled trapezoid, 8×8 stipple pattern.
-ostrap100 Fill 100×100 opaque stippled trapezoid, 8×8 stipple pattern.
-tiletrap10 Fill 10×10 tiled trapezoid, 4×4 tile pattern.
-tiletrap100 Fill 100×100 tiled trapezoid, 4×4 tile pattern.
-bigstrap10 Fill 10×10 transparent stippled trapezoid, 161×145 stipple pattern.
-bigstrap100 Fill 100×100 transparent stippled trapezoid, 161×145 stipple pattern.
-bigostrap10 Fill 10×10 opaque stippled trapezoid, 161×145 stipple pattern.
-bigostrap100 Fill 100×100 opaque stippled trapezoid, 161×145 stipple pattern.
-bigtiletrap10 Fill 10×10 tiled trapezoid, 161×145 tile pattern.
-bigtiletrap100 Fill 100×100 tiled trapezoid, 161×145 tile pattern.
-eschertiletrap10
Fill 10×10 tiled trapezoid, 216×208 tile pattern.
-eschertiletrap100
Fill 100×100 tiled trapezoid, 216×208 tile pattern.
-complex10 Fill 10-pixel/side complex polygon.
-complex100 Fill 100-pixel/side complex polygon.
-ftext Draw a character in 80-character line (6×13).
-f8text Draw a character in 70-character line (8×13).
-f9text Draw a character in 60-character line (9×15).
-f14text16 Draw a two-byte character in 40-character line (k14).
-tr10text Draw a character in 80-character line (Times-Roman 10).
-tr24text Draw a character in 30-character line (Times-Roman 24).
-polytext Draw a character in a 20/40/20 line (6×13, Times-Roman 10, 6×13).
-fitext Draw a character in a 80-character image line (6×13).
-f8itext Draw a character in a 70-character image line (8×13).
-f9itext Draw a character in a 60-character image line (9×15).
-f14itext16 Draw a two-byte character in a 40-character image line (k14).
-tr10itext Draw a character in a 80-character image line (Times-Roman 10).
-tr24itext Draw a character in a 30-character image line (Times-Roman 24).
-scroll10 Scroll 10×10 pixels vertically.
-scroll100 Scroll 100×100 pixels vertically.
-scroll500 Scroll 500×500 pixels vertically.
-copywinwin10
Copy a 10×10 square from window to window.
-copywinwin100
Copy a 100×100 square from window to window.
-copywinwin500
Copy a 500×500 square from window to window.
-copypixwin10 Copy a 10×10 square from pixmap to window.
-copypixwin100
Copy a 100×100 square from pixmap to window.
-copypixwin500
Copy a 500×500 square from pixmap to window.
-copywinpix10 Copy a 10×10 square from window to pixmap.
-copywinpix100
Copy a 100×100 square from window to pixmap.
-copywinpix500
Copy a 500×500 square from window to pixmap.
-copypixpix10 Copy a 10×10 square from pixmap to pixmap.
-copypixpix100
Copy a 100×100 square from pixmap to pixmap.
-copypixpix500
Copy a 500×500 square from pixmap to pixmap.
-copyplane10 Copy a 10×10 1-bit deep plane.
-copyplane100 Copy a 100×100 1-bit deep plane.
-copyplane500 Copy a 500×500 1-bit deep plane.
-putimage10 PutImage 10×10 square.

LEXICON

-putimage100 PutImage 100×100 square.
-putimage500 PutImage 500×500 square.
-shmput10 PutImage 10×10 square, MIT shared memory extension.
-shmput100 PutImage 100×100 square, MIT shared memory extension.
-shmput500 PutImage 500×500 square, MIT shared memory extension.
-getimage10 GetImage 10×10 square.
-getimage100 GetImage 100×100 square.
-getimage500 GetImage 500×500 square.
-noop X protocol NoOperation.
-atom GetAtomName.
-prop GetProperty.
-gc Change graphics context.
-create Create child window and map using MapSubwindows.
-ucreate Create unmapped window.
-map Map child window via MapWindow on parent.
-unmap Unmap child window via UnmapWindow on parent.
-destroy Destroy child window via DestroyWindow parent.
-popup Hide/expose window via Map/Unmap popup window.
-move Move window.
-umove Moved unmapped window.
-movetree Move window via MoveWindow on parent.
-resize Resize window.
-uresize Resize unmapped window.
-circulate Circulate lowest window to top.
-ucirculate Circulate unmapped window to top.

X Defaults

This program uses no X defaults.

See Also

X, x11perfcomp, xbench, X utilities

Notes

x11perf is mostly the responsibility of Joel McCormack. It is based upon the **x11perf** developed by Phil Karlton, Susan Angebranndt, and Chris Kent, who wished to assess performance differences between various servers. For a general release to the world, **x11perf** was rewritten to ease making comparisons between widely varying machines, to cover most important (and unimportant) X functionality, and to exercise graphics operations in as many different orientations and alignments as possible.

x11perfcomp — X Utility

Compare the output of multiple runs of x11perf
x11perfcomp [-r -ro] [-l *labelfile***]** *label_file datafile1 ... datafileN*

The script **x11perfcomp** tabulates the output of multiple runs of the command **x11perf**. This lets you easily compare the performance of different servers, or of the same server under various operating conditions.

Each *datafile* holds the output of a run of **x11perf**. *labelfile* holds labels that identify the tests that **x11perf** executed. This file must precede any *datafile* which holds the output of an **X11perf** run. To generate such a file, use **x11perf**'s option **-label**, as described in its Lexicon entry.

x11perfcomp recognizes the following command-line options:

-r Include relative rates in the output.

-ro Report only relative rates.

-l *labelfile*
Use *labelfile* as the label file. The default is to use the first file named on the command line.

See Also

X, x11perf, X utilities

Notes

An 80-column line can hold the data output by up to four runs of **x11perf**. For information on how to print longer lines, see the entries in the COHERENT Lexicon for **pr** and **prps**.

x11perfcomp was written by Mark Moraes of the University of Toronto (moraes@csri.toronto.edu) and Joel McCormack of DEC Western Research Laboratory (joel@decwrl.dec.com).

xauth — X Utility

Display/edit authorization information
xauth [-biqv -f *authfile***] [***command* **[arguments]]**

The X utility **xauth** lets you display and edit the authorization information with which users connect to the X server. It is used most commonly to extract authorization records from one machine and merge them into the records on another, such as when using remote logins or granting authorization to other users. Each *command* given below can be entered interactively, entered on **xauth**'s command line, or embedded within a script.

xauth recognizes the following options:

-b Do not break any authority file locks before proceeding. With this option, **xauth** is used only to clean up stale locks.

-f *authfile*
Use the authority file *authfile*. By default, **xauth** uses the authority file named by the environmental variable **XAUTHORITY**, or the file **$HOME/.Xauthority**.

-i Ignore all locks on authority files. Normally, **xauth** refuses to read or edit any authority files that have been locked by other programs.

-q Quiet: Do not print unsolicited messages about authority status. This is the default if a command is given on **xauth**'s command line or if the standard output is not directed to a terminal.

-v Verbose: Print a message to indicate the outcome of each operation. This is the default option if **xauth** is reading commands from the standard input and its standard output is directed to a terminal.

Commands

As noted above, **xauth** does its work by executing one or more commands. If **xauth** finds no command on its command line, it reads the standard input and executes all commands it received until it receives the command **exit**, receives the command **quit**, or receives EOF (**<ctrl-D>** under COHERENT).

xauth recognizes the following commands:

? Print a summary of all commands.

add *display protocol hexkey*
Add an authorization entry for *display*. *protocol* gives the protocol that *display* uses. The *protocol* of '.' is an abbreviation for the protocol **MIT-MAGIC-COOKIE-1**. *hexkey* gives the key data, in the form of a string of pairs of hexadecimal numerals; the first numeral gives the most significant four bits and the second the least significant four bits.

exit Write out the modified authorization file, and exit from **xauth**. EOF is equivalent to this command.

extract *file display* **[...** *display***]**
Write the authorization entries for each *display* into *file*. The *file* of '-' indicates the standard output. This command can be used with the **xauth**'s command **merge** to copy permissions from one authorization into another.

help [*string***]**
Print help information about all commands that begin with *string*. If no *string* is given, list information about all commands.

info Print a summary of information about the authorization file itself.

list [*display* **[** *... display***]]**
Print onto the standard output the authorization information for each *display*. If no *display* is named, print information about all displays. Key data are printed in the hexadecimal format used by the **add** command, described above.

merge [*file* **[** *... file***]]**
Merge the authorization information from each *file* into the authorization file. The *file* '-' indicates the standard input.

nextract *file display* **[...** *display***]**
Same as **extract**, described above, except that the authorization information is written in a numeric format suitable for non-binary transmission (such as secure electronic mail).

nlist [*display* **[** *... display***]]**
Like the command **list**, described above, print output in the numeric form output by the command **nextract**, described above.

nmerge [*file* **[** *... file***]]**
Same as merge, except that the information in each *file* is in the numeric form output by the command **nextract** (described above).

quit Exit from **xauth**, and do not modify the authorization file. The interrupt character (**<ctrl-C>** under COHERENT) is equivalent to this command.

remove *display* **[...** *display***]**
Remove from the authorization file the information for each *display*.

source *file*
Read *file* and execute all of the **xauth** commands it contains. **xauth** ignores all lines within *file* that begin with a pound sign '#'. The *file* '-' indicates the standard input.

Display Names

Display names used with the **xauth** commands **add**, **extract**, **list**, **merge**, **nextract**, **nlist**, **nmerge**, and **remove** use the same format as the environmental variable **DISPLAY** and the common command-line argument **-display**. **xauth** ignores display-specific information (such as the screen number), because it is unnecessary. Same-machine connections (such as local-host sockets, shared memory, and the Internet Protocol host name **localhost**) are referred to as *hostname***/unix:***displaynumber* so that local entries for different machines can be stored in one authority file.

Environment

xauth reads the following environmental variables:

XAUTHORITY
The name of the authority file to use if the option **-f** is not used. If this variable is not set, **xauth** reads the contents of file **$HOME/.Xauthority**.

HOME The user's home directory if **XAUTHORITY** is not defined.

See Also

xdm, X utilities

Notes

xauth does not interact with the X server.

If your network is not secure, be sure to encrypt the authorization information before you pass it to another machine. Likewise, the protocol **MIT-MAGIC-COOKIE-1** is not very useful in insecure environments. Sites that are interested in additional security should consider using encrypted authorization mechanisms, such as Kerberos.

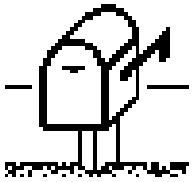
Each *display* option in the above commands uses the same format as the environmental variable **DISPLAY**.

xauth was written by Jim Fulton of the MIT X Consortium.

xbiff — X Client

Notify the user that mail has arrived
xbiff [*options***]**

The X client **xbiff** automatically notifies you when you have mail. It displays the following picture of an old-fashioned mailbox:



When you do not have mail, the mailbox's "flag" is down; when new mail arrives, the "flag" pops up, the image is redrawn in reverse video, and the terminal beeps. Clicking on the "mailbox" resets its "flag" to the down position and redraws the image in normal video.

Options

xbiff recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host*[**:***server*][**.***screen*]
Display the window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-file *mailfile*
The user's mailbox file is *mailfile*. **xbiff** watches *mailfile* and pops its "flag" whenever *mailfile*'s status changes. By default, **xbiff** watches **/usr/spool/mail/***user*, where *user* is the login identifier of the user who is running **xbiff**.

-geometry *geometry*
Set the geometry of the window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-help Print a brief message that describes **xbiff** and its options.

-rv Simulate reverse video by swapping the foreground and background colors.

-shape Shape the mailbox window, should masks for empty or full images be available.

update *seconds*
Wait *seconds* before checking the mailbox and, if necessary, updating the display. The default is 60 seconds.

volume *percent*
Indicate how loudly **xbiff** should ring the bell when new mail arrived, as a percent of the terminal's maximum volume. The default is 33%.

Resources

xbiff has the application class name **XBiff**. It uses the **Mailbox** widget in the X Toolkit. It understands all of the core resource names and classes, plus the following:

checkCommand(class **CheckCommand)**
Name a shell command to execute in order to check the status of the mailbox, rather than checking the size of the mail file. The command string is passed as an argument to the function **system()** (which is described in the COHERENT Lexicon), and so may contain redirection. **xbiff** assumes that a return value of zero indicates that new mail is waiting, one indicates that there is no change in size, and two that mail has been cleared.

emptyPixmap(class **Pixmap)**
Name the bit map to display when no new mail is present.

emptyPixmapMask(class **PixmapMask)**
Name the mask for the bit map to display when no new mail is present.

file(class **File)**
Specify the name of the file to monitor. Setting this option is the same as using the **-file** option described above.

flip(class **Flip)**
Invert the image when new mail arrives. The default is true.

foreground(class **Foreground)**
Give the color of the foreground. The default is white.

fullPixmap(class **Pixmap)**
Name the bit map to display when mail arrives.

fullPixmapMask(class **PixmapMask)**
Name the mask for the bit map to display when mail arrives.

height(class **Height)**
Give the height of the mailbox.

onceOnly(class **Boolean)**
Ring the bell only once when new mail arrives. Do not ring the bell when new mail arrives thereafter until at least one interval has passed, the user has clicked on the mailbox, or read his mail.

reverseVideo(class **ReverseVideo)**
Invert the foreground and background colors to indicate reverse video. The same as the option **-rt**, described above.

shapeWindow(class **ShapeWindow)**
Specify whether to shape the window to the **fullPixmapMask** and **emptyPixmapMask**. The default is false. Same as the option **-shape**.

update(class **Interval)**
The interval, in seconds, that **xbiff** waits until it checks the mailbox. The default is 60 seconds. Same as the option **-update**.

volume(class **Volume)**
Specify how loudly the bell should be rung, as a percent of maximum volume. The default is 33%. This is the same as the option **-volume**.

width(class **Width)**
Specify the width of the mailbox.

Actions

The Mailbox widget provides the following action for use in event translations:

check() Check for new mail and display the flag appropriately.

unset() Display the "flag" to the lowered position until new mail arrives.

set() Display the flag in the raised position until the user clicks on the mailbox.

The default translation is:

```
<ButtonPress>:  unset()
```

Environment

xbiff reads the following environmental variables:

DISPLAY
The default host and display.

XENVIRONMENT
The name of the resource file that overrides the global resources stored in the property **RESOURCE_MANAGER**.

See Also

X clients, xrdb

Notes

xbiff was written by Jim Fulton of the MIT X Consortium, and Ralph R. Swick of DEC/MIT Project Athena.

xcalc — X Client

Scientific calculator for X
xcalc [-stipple] [-rpn] [-*toolkitoption...***]**

The X client **xcalc** emulates the Texas Instruments (TI) 30 and Hewlett-Packard (HP) 10C scientific calculators.

xcalc recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rpn Use reverse Polish notation. In this mode, the calculator looks and behaves like an HP-10C. Without this flag, it emulates a TI-30.

-rv Simulate reverse video by exchanging the foreground and background colors.

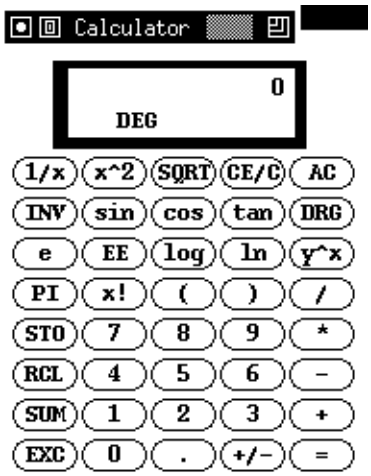
-stipple Draw the background of the calculator as a stipple of the foreground and background colors. On monochrome displays, this improves the appearance.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Key Operations

When you invoke **xcalc**, it displays a picture of a scientific calculator on your screen — of an HP-10C if you use the command-line option **-rpn**, or of a TI-30 if you did not. To operate the virtual calculator, move the mouse cursor to the key you wish to press, and then punch the left mouse button. Some operations can also be invoked by pressing appropriate keys on the keyboard. To quit, move the mouse cursor to key `(AC)` (on the TI calculator) or the key `(ON)` (on the TI calculator), and press the right mouse button.

The following shows the TI calculator, which is the default:



The following two sections describe the two virtual calculators in detail.

Calculator Keys — TI Mode

The number keys and the keys

```
(+/-) (+) (-) (*) (/) (=)
```

do exactly what you expect them to do. The operators obey the standard rules of precedence. Thus, pressing the key sequence

```
(3) (+) (4) (*) (5) (=)
```

produces 23, not 35. You can use parentheses to override this.

You can select the entire number in the calculator's display, to paste the result of a calculation into text.

The following describes what happens when you click the function keys on the virtual calculator. These are useful if you are interested in defining a customized calculator.

`(0)` through `(9)`
: Each digit key is linked to function **digit(***n***)**, where *n* is the corresponding digit, 0 through 9.

`(1/x)` Replace the number in the display with its reciprocal. The corresponding procedure is **reciprocal()**.

`(x^2)` Square the number in the display. The corresponding procedure is **square()**.

`(SQRT)` Compute the square root of the number in the display. The corresponding procedure is **squareRoot()**.

`(CE/C)` When pressed once, clear the number in the display without clearing the state of the machine. This lets you re-enter a number if you make a mistake. Pressing it twice also clears the state. The corresponding procedure for TI mode is **clear()**.

`(AC)` Clear the display, the state, and the memory. Pressing it with the right mouse button "turns off" the calculator, which exits you from the program. The action procedure to clear the state is **off()**; to quit, **quit()**.

`(INV)` Invert function. See the individual function keys for details. The corresponding procedure is **inverse()**.

`(sin)` Compute the sine of the number in the display, as interpreted by the current DRG mode (see the description of the key `(DRG)`,below). If inverted, it computes the arcsine. The corresponding procedure is **sine()**.

`(cos)` Computes the cosine, or arccosine when inverted. The corresponding procedure is **cosine()**.

`(tan)` Compute the tangent, or arctangent when inverted. The corresponding procedure is **tangent()**.

`(DRG)` Change the DRG mode, as indicated by the fields **DEG**, **RAD**, or **GRAD** at the bottom of of the calculator "liquid crystal" display. When in **DEG** mode, numbers in the display are taken as being degrees. In **RAD** mode, numbers are in radians. In **GRAD** mode, numbers are in grads. When inverted, the **DRG** key converts degrees to radians to grads.

For example, to put the calculator into **DEG** mode, press the key sequence:

```
(4) (5) (INV)(DRG)
```

The display shows **.78539816**, which is 45° converted to radians. The corresponding procedure is **degree()**.

(e) The constant *e*. The corresponding procedure is **e()**.

(EE) Enter exponential numbers. For example, to get **-2.3E-4**, press the key sequence:

```
(2) (•) (3) (+) (EE) (4) (+)
```

The corresponding procedure is **scientific()**.

(log) Calculates the logarithm (base 10) of the number in the display. When inverted, it raises 10.0 to the power of the number in the display. For example, entering the key sequence

```
(3) (INV)(log)
```

1000 appears on the display. The corresponding procedure is **logarithm()**.

(ln) Calculate the natural logarithm (base *e*) of the number in the display. When inverted, it raises *e* to the number in the display. For example, pressing the key sequence

```
(e) (ln)
```

shows 1 in the display. The corresponding procedure is **naturalLog()**.

(y^x) Raise the number on the left to the power of the number on the right. For example, pressing the key sequence:

```
(2) (y^x)(3) (=)
```

results in 8, which is two raised to the third power. For a further example, the key sequence:

```
(«) (1) (+) (2) (+) (3) (») (y^x)(«) (1) (+) (2) (») (=)
```

equals

```
(6) (y^x)(3)
```

which equals 216. The corresponding procedure is **power()**.

(PI) The constant π. The corresponding procedure is **pi()**.

(x!) Compute the factorial of the number in the display. The number in the display must be an integer in the range 0 through 500 — although, depending on whether you have a mathematics coprocessor, **xcalc** might overflow long before that. The corresponding procedure is **factorial()**.

(«) Left parenthesis. The corresponding procedure for TI calculators is **leftParen()**.

(») Right parenthesis. The corresponding procedure for TI calculators is **rightParen()**.

(/) Division. The corresponding procedure is **divide()**.

(*) Multiplication. The corresponding procedure is **multiply()**.

(-) Subtraction. The corresponding procedure is **subtract()**.

(+) Addition. The corresponding procedure is **add()**.

(STO) Store: copy the number in the display in system memory. The corresponding procedure is **store()**.

(RCL) Recall: copy the number from memory to the display. The corresponding procedure is **recall()**.

(SUM) Add the number in the display to the number in memory. The corresponding procedure is **sum()**.

(EXC) Swap the number in the display with the number in memory. The corresponding procedure for the TI calculator is **exchange()**.

(.) Decimal point. The action procedure is **decimal()**.

LEXICON

(+/-) Negate: change sign. The corresponding procedure is **negate()**.

(=) Perform calculation. The TI-specific action procedure is **equal()**.

Calculator Keys — HP Mode

The number keys and the keys

(CHS) (+) (-) (*) (/) (=) (ENTR)

do exactly what you expect them to do. Many of the remaining keys are the same as in TI mode. The differences are detailed below.

(ENTR) Enter: execute the key sequence just entered. The corresponding procedure is **enter()**.

(<-) Backspace: erase from the display the last number key pressed. Inverse backspace clear the X register. The corresponding procedure is **back()**.

(ON) Clear the display, the state, and the memory. Clicking it with the right mouse button turns off the calculator. To clear the state of the virtual calculator, the procedure is **off()**; to quit, **quit()**.

(INV) Invert the meaning of the function keys. This would be the (f) key on an HP calculator, but **xcalc** does not display multiple legends on each key. See the individual function keys for details.

(10^x) Raise 10.0 to power of the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display. The corresponding procedure is **tenpower()**.

(e^x) Raise *e* to the number in the top of the stack. When inverted, **xcalc** calculates the log (base *e*) of the number in the display. The action procedure is **epower()**.

(STO) Copy the number in the top of the stack to a memory location. There are ten memory locations. The desired memory is specified by following this key with a digit key.

(RCL) Push the number from the specified memory location onto the stack.

(SUM) Add the number on top of the stack to the number in the specified memory location.

(x:y) Exchange the numbers in the top two stack positions, the X and Y registers. The corresponding procedure is **XexchangeY()**.

(Rv) Roll the stack downward. When inverted, it rolls the stack upward. The corresponding procedure is **roll()**.

Blank keys represent programming functions on the HP-10C whose functionality has not been duplicated in **xcalc**.

Finally, there are two additional action procedures: **bell()**, which rings the bell; and **selection()**, which performs a cut on the entire number in the calculator's "liquid crystal" display.

Keyboard Accelerators

Accelerators are shortcuts for entering commands. By pressing one key on your keyboard, you can invoke an **xcalc** function that might require clicking several keys on the virtual calculator. Even though the word "calculator" is descended from the Greek word for "pebble," you may prefer not to have to program computations by shoving a rock around on your desk.

xcalc provides some sample keyboard accelerators; you can also customize accelerators on your own. The numeric keypad accelerators provided by **xcalc** should be intuitively correct. The following gives the accelerators that **xcalc** defines:

TI Key	*HP Key*	*Keyboard Accelerator*	*TI Function*	*HP Function*
(SQRT)	(SQRT)	(r)	**squareRoot()**	**squareRoot()**
(AC)	(ON)	(space)	**clear()**	**clear()**
(AC)	(<-)	(Del)	**clear()**	**back()**
(AC)	(<-)	(Backspace)	**clear()**	**back()**
(AC)	(<-)	(ctrl-H)	**clear()**	**back()**

(AC)		(Clear)	**clear()**	
(AC)	(ON)	(q)	**quit()**	**quit()**
(AC)	(ON)	(ctrl-C)	**quit()**	**quit()**
(INV)	(i)	(i)	**inverse()**	**inverse()**
(sin)	(s)	(s)	**sine()**	**sine()**
(cos)	(c)	(c)	**cosine()**	**cosine()**
(tan)	(t)	(t)	**tangent()**	**tangent()**
(DRG)	(DRG)	(d)	**degree()**	**degree()**
(e)		(e)	**e()**	
(ln)	(ln)	(l)	**naturalLog()**	**naturalLog()**
(y^x)	(y^x)	(^)	**power()**	**power()**
(PI)	(PI)	(p)	**pi()**	**pi()**
(x!)	(x!)	(!)	**factorial()**	**factorial()**
(«)		(«)	**leftParen()**	
(»)		(»)	**rightParen()**	
(/)	(/)	(/)	**divide()**	**divide()**
(*)	(*)	(*)	**multiply()**	**multiply()**
(-)	(-)	(-)	**subtract()**	**subtract()**
(+)	(+)	(+)	**add()**	**add()**
(=)		(=)	**equal()**	
(0)	(0)	(0)	**digit()**	**digit()**
(1)	(1)	(1)	**digit()**	**digit()**
(2)	(2)	(2)	**digit()**	**digit()**
(3)	(3)	(3)	**digit()**	**digit()**
(4)	(4)	(4)	**digit()**	**digit()**
(5)	(5)	(5)	**digit()**	**digit()**
(6)	(6)	(6)	**digit()**	**digit()**
(7)	(7)	(7)	**digit()**	**digit()**
(8)	(8)	(8)	**digit()**	**digit()**
(9)	(9)	(9)	**digit()**	**digit()**
(.)	(.)	(.)	**decimal()**	**decimal()**
(+/-)	(CHS)	(n)	**negate()**	**negate()**
	(x:y)	(x)		**XexchangeY()**
	(ENTR)	(¢)		**enter()**
	(ENTR)	(ctrl-J)		**enter()**

Customization

The application class name is **XCalc**.

xcalc has an enormous application defaults file that specifies the position, label, and function of each key on the calculator, and gives translations to serve as keyboard accelerators. Because these resources are not specified in the source code, you can create a customized calculator by writing a private application defaults file, using the Athena Command and Form widget resources to specify the size and position of buttons, the label for each button, and the function of each button.

You can specify the foreground and background colors of each calculator key. For the TI calculator, a classical color resource specification resembles:

```
XCalc.ti.Command.background:  gray50
XCalc.ti.Command.foreground:  white
```

For each of buttons 20, 25, 30, 35, and 40, specify:

```
XCalc.ti.button20.background:  black
XCalc.ti.button20.foreground:  white
```

For each of buttons 22, 23, 24, 27, 28, 29, 32, 33, 34, 37, 38, and 39:

LEXICON

```
XCalc.ti.button22.background:  white
XCalc.ti.button22.foreground:  black
```

Application Resources

The following lists the resources used by **xcalc**:

rpn (Class **Rpn)**
Use **rpn** mode. The default is TI mode. Same as the command-line option **-rpn**.

stipple (Class **Stipple)**
Stipple the background. The default is **on** for monochrome displays, and **off** for color displays. Same as the command-line option **-stipple**.

cursor (Class **Cursor)**
The name of the symbol used to represent the pointer. The default is **hand2**.

Colors

If you want **xcalc** to use its TI color palette, include the following in the **#ifdef COLOR** section of the file you read with **xrdb**:

```
*customization:                -color
```

This tells **xcalc** to pick up the colors in the application-defaults color customization file **/usr/X11/lib/app-defaults/XCalc-color**.

See Also

X clients

Notes

xcalc was written by John Bradley of the University of Pennsylvania, Mark Rosenstein of MIT Project Athena, and Donna Converse of the MIT X Consortium.

xclipboard — X Utility

Hold multiple selections for later retrieval
xclipboard [-w -nw -display [*host***]:***server***[.***screen***] -geometry** *geometry***]**

By default, the X system lets you cut only one "hunk" of text at a time. The X utility **xclipboard** lets you store multiple hunks of text on an internal "clipboard". You can display these hunks one at a time, and select one or another to copy into another window.

Using xclipboard

When you invoke **xclipboard**, it draws the following window:

Its buttons do the following:

`(Quit)` Exit from **xclipboard**.

`(Delete)`
Delete the current buffer and display the next.

`(New)` Create a new text buffer.

`(Save)` Save the current buffer into a file. When you select this option, **xclipboard** displays a pop-up window to prompt you for the name of the file to save the cut text.

`(Next)` Display the next buffer.

`(Prev)` Display the previous buffer.

The window also shows a small sub-window, within which text is displayed.

To copy a hunk of text to the clipboard, do the following:

- Click the button (New)to open a new entry on the clipboard.
- Move the mouse cursor to the text you wish to copy; then press the left mouse button and drag the mouse cursor across the text. X redisplays the text in reverse video, to show that it has been cut.
- Move the mouse pointer into the clipboard window's text sub-window. Press the middle mouse button: this drops the cut text into the clipboard.
- Finally, move the mouse pointer back into the window from which you cut the text and click the left mouse button. This un-highlights the cut text, to indicate that it is no longer cut.

To copy text from the clipboard back into an application, do the following:

- Click on the (Next)and (Prev)buttons until **xclipboard** displays in its text sub-window the hunk of text you want.
- Press the left mouse button and sweep the mouse cursor across the text you wish to copy. Again, the text is highlighted.
- Move the mouse cursor into the window into which you wish to copy the text. Click the center mouse button. This drops the cut text into place.
- Finally, move the mouse cursor into text sub-window of **xclipboard**'s window, and click the left mouse button again. This un-highlights the text, to show that it is no longer cut.

Options

xclipboard recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display its window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-nw Do not wrap long lines of text. This is the default.

-rv Simulate reverse video by exchanging the foreground and background colors.

-w Wrap onto the next line each line of text that is too long to fit onto one line.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Selection

xclipboard stores each hunk of text as a string. You can select each hunk separately from other hunks. Text is copied to the clipboard whenever a client asserts ownership of the CLIPBOARD selection. Text is copied from the clipboard whenever a client requests the contents of the CLIPBOARD selection. The following gives examples of event bindings that you may wish to include in the resource file of a client that uses the clipboard: in this file:

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

Each time an application asserts the clipboard, **xclipboard** transfers the text that application selected into a new buffer and displays it in its text window. **xclipboard** does not automatically delete these buffers: you must use the delete button on the mouse to drop useless buffers into the bit bucket.

xclipboard also responds to requests for the clipboard selection from other clients by sending the entire contents of the currently displayed buffer.

LEXICON

Resources

xclipboard accepts all of the standard X Toolkit resource names and classes. In addition, it accepts:

wordWrap(class **WordWrap)**
Specify whether long lines of text should wrap to the following lines. The default is no.

Files

/usr/X11/lib/app-defaults/XClipboard — Resource file

See Also

X utilities

Notes

xclipboard was written by Ralph R. Swick of DEC/MIT Project Athena, and by Chris Peterson and Keith Packard of the MIT X Consortium.

xclock — X Client

Display a clock
xclock [*options***]**

xclock displays a clock that continually displays the current time. This clock can either be analogue (that is, an old-fashioned clock with hands) or digital. The latter shows the hour and minute in 12-hour format; the latter gives hours and minutes in 24-hour (military) format, plus the day, month, and year. The clock is initialized to the time on your system.

The standard clock is an analogue clock with a white background and black foreground:

The following command-line options let you customize your clock:

-analog Display an analogue clock. This is the default.

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-chime Chime (beep) once on the half hour and twice on the hour.

-digital Display a digital clock that gives the date as well as the time.

-display *host***[:***server***][.***screen***]**
Display the clock on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-geometry *geometry*
Set the geometry of the clock to *geometry*. The term "geometry" means the dimensions of the clock and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-hd *color* Set the color of the analogue clock's hands to *color*. The default is black.

-help Display a brief summary of **xclock**'s syntax and options.

-hl *color* Set the color of the edge of the analogue clock's hands to *color*. The default is black.

-padding *pixels*
Set to *pixels* the padding between the window border and the edge of the clock. The default is ten pixels for the digital clock, and eight for the analogue.

-rv Simulate reverse video by exchanging the foreground and background colors.

-update *seconds*
Wait *seconds* before updating the time displayed on the clock. The default is 60 seconds. If set to less than 30 seconds, the analogue clock also displays a second hand. **xclock** automatically updates the time whenever the clock is iconified and then re-exposed.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Resources

xclock uses the Athena Clock widget. It understands all of the core resource names as classes. In addition, it understands the following:

analogue(class **Boolean)**
Indicate whether to use an analogue clock. The default is true.

background(class **Background)**
Give the color of the foreground. The default is white.

chime(class **Boolean)**
Indicate whether to chime on the hour and half-hour. The default is false.

font(class **Font)**
Give the font to use with the digital clock. Variable-width fonts are not always displayed correctly.

foreground(class **Foreground)**
Give the color of the foreground. The default is white.

hands(class **Foreground)**
Give the color of the inside of the analogue clock's hands. The default is the foreground color.

height(class **Height)**
Give the height of the mailbox.

highlight(class **Foreground)**
Give the color used to highlight the analogue clock's hands. The default is the foreground color.

padding(class **Margin)**
Give the amount of padding, in pixels, to display between the border of the window and the edge of the clock. The default is eight.

reverseVideo(class **ReverseVideo)**
Indicate reverse video by exchanging the foreground and background colors.

update(class **Interval)**
The interval, in seconds, that **xclock** waits until it updates the time on the clock. The default is 60 seconds. Same as the option **-update**.

width(class **Width)**
Specify the width of the mailbox.

Example

The command

```
xclock -digital -update 1 -foreground red -chime
```

displays a digital clock that shows the current date and time in red characters. The clock is updated every second, and chimes on the hour and half-hour.

LEXICON

Environment

xclock reads the following environmental variables:

DISPLAY
The default host and display.

XENVIRONMENT
The name of the resource file that overrides the global resources stored in the property **RESOURCE_MANAGER**.

Files

/usr/X11/lib/app-defaults/XClock — Default resources.

See Also

oclock, X clients

Notes

xclock was written by Tony Della Fera and Dave Mankins of MIT Project Athena, and Edward Moy of the University of California, Berkeley.

There is no way to turn the clock off. You can, however, invoke **xkill** to kill it.

xcmsdb — X Utility

Manipulate xlib screen-color characterization data
xcmsdb [-color] [-format [32|16|8]] [-query] [-remove] [*filename***]**

xcmsdb can load, query, or remove the Screen Color Characterization Data (SCCD) stored within a property on the screen's root window. The SCCD are an integral part of **xlib**; applications need them to convert properly between device-independent and device-dependent color specifications.

xlib uses the following properties:

XDCCC_LINEAR_RGB_MATRICES
XDCCC_LINEAR_RGB_CORRECTION
Store color-characterization data for color monitors.

XDCCC_GRAY_SCREENWWHITEPOINT
XDCCC_GRAY_CORRECTION
Store data for gray-scale monitors.

Because **xlib** allows the addition of Screen Color Characterization Function Sets, an added function sets may place its SCCD onto other properties. **xcmsdb** is unaware of these other properties; therefore, you must use a similar utility provided with the function set, or use the utility **xprop**.

xcmsdb transforms the contents of *filename* (or the standard input if the command line has no *filename*) for storage within properties, provided its command line does not specify the options **-query** or **-remove**.

xcmsdb recognizes the following command-line options:

-color Limit the options **-query** and **-remove** (described below) so that they check only for the properties **XDCCC_LINEAR_RGB_MATRICES** and **XDCCC_LINEAR_RGB_CORRECTION**. If you do not set this option, options **-query** and **-remove** check for all properties.

-format [32|16|8]
Specify the format for the property **XDCCC_LINEAR_RGB_CORRECTION**. Flags **32**, **16**, and **8** set the number of bits per entry; the more bits, the greater the precision of encoded floating-point values. The default is 32 bits per entry.

-query Read the **XDCCC** properties from the screen's root window. If successful, **xcmsdb** transforms the data into a more readable format, then writes them to the standard-output device.

-remove
Remove the **XDCCC** properties from the screen's root window.

Environment

xcmsdb reads the environmental variable **DISPLAY** to find the display and screen to use.

See Also

xprop, X utilities

Notes

xcmsdb was written by Chuck Adams of Tektronix, Inc.

xcmstest — X Utility

XCMS test program
xcmstest [-echo] [-display *displayname***]**

xcmstest is a simple interface for testing the XCMS API library.

Before Invocation

Ensure that the Screen Color Characterization Data (SCCD) has been placed into the appropriate properties on the screen's root window. To load these properties, refer to the entry in this manual for the utility **xcmsdb**.

xcmstest recognizes the following command-line options:

-echo Echo a command as it is executed. This option usually is used when reading a file of commands via the standard input.

-display *displayname*
Read the display *displayname*.

To convert color names using a client-side color name data base, the data-base file must be specified via the environment variable **XCMSDB**. To use the sample database provided with this release, set **XCMSDB** as follows:

```
setenv XCMSDB ‘pwd‘/Xcms.txt
```

Using xcmstest

Upon invocation, you should receive the prompt

```
XCMS >
```

At this point you can enter commands. **xcmstest** recognizes the following commands:

init Initiate contact with *displayname*.
quit Exit from **cmstest**.
bye Synonym for **quit**.
halt Synonym for **quit**.
q Synonym for **quit**.
list List available commands
? Synonym for **list**.
CreateColormap *name AllocFlag*
AllocFlag must be either **AllocNone** or **AllocAll**.
FreeColormap
Free the current color map.
Set_idir *dirname*
Set_vdir *dirname*
Set_rdir *dirname*
Get_idir
Get_vdir
Get_rdir
XSynchronize ON|OFF
AllocColor *Format Stim1 Stim2 Stim3 Result_Format* **[***Colormap***]**
AllocNamedColor *color_string result_format* **[***Colormap***]**
ConvertColor *From_Format Stim1 Stim2 Stim3 To_Format*
LookupColor *color_string result_format* **[***Colormap***]**

QueryColor *Pixel result_format* **[***Colormap***]**
QueryColors *result_format* **[***Colormap***]**
QueryColors *result_format* **[***Colormap***]**
StoreColor <Pixel> <Format> <Stim1> <Stim2> <Stim3> [Colormap]
StoreColors [*Colormap***]**
AddDIColorSpace *Format*
Format must be one of **CIELab**, **CIELuv**, or **TekHVC**.
FormatOfPrefix *prefix*
PrefixOfId *Format*
Format must be one of **UNDEFINED**, **CIEXYZ**, **CIExyY**, **CIEuvY**, **CIELab**, **CIELuv**, **TekHVC**, **RGBi**, or **RGB**.
MaxChroma *Hue Value*
MaxValue *Hue Chroma*
MaxValueSamples *Hue nSamples*
MaxValueChroma *Hue*
MinValue *Hue Chroma*
AdjustValue *Hue Value Chroma*
ReduceChroma *Hue Value Chroma*
ShortestValueChroma *Hue Value Chroma*
ShortestValueChroma *Hue Value Chroma*
XAllocNamedColor *color_string* **[***Colormap***]**
XLookupColor *color_string* **[***Colormap***]**
XParseColor *color_string* **[***colormap***]**
XStoreNamedColor *color_string pixel* **[***Colormap***]**

You can, if you wish, redirect the standard input so that **xcmstest** can read and execute a file of commands. In this case, you should use the option **-echo**, which tells **xcmstest** to echo the command before executing it. This creates a more readable output.

See Also

xcmsdb, X utilities

Notes

Color-name strings passed to **xcmstest** commands cannot contain a space. However, because spaces are ignored by the **XCMS API** library, you can pass a color name with the spaces eliminated. For example:

```
LookupColor cornflowerblue UNDEFINED RGB
```


xcutsel — X Utility

Copy text between the cut buffer and the primary selection
xcutsel [-*toolkitoption ...***] [-selection** *selection***] [-cutbuffer** *number***]**

Prior to release 3 of the X Window System, text that you cut with the mouse was copied into the system's cut buffer. Beginning with release 3, X added the *primary selection* — a special buffer that holds the text highlighted with the mouse. (X stores the primary selection in the property **PRIMARY**. You can use the utility **xprop** to to examine the contents of this property.) Release 3 added the primary selection because some applications perform transformations on the text kept in the cut buffer; it set aside the primary selection to keep the "raw" text available for pasting into other windows.

Thus, when you cut text, X copies the cut text into both the copy buffer and the primary selection. When you paste text, X copies the primary selection into the current window; if the primary selection is empty for some reason, it copies the text that is in the cut buffer.

This creates a problem, however, because X applications written prior to release 3 use the cut buffer for cutting and pasting, and do not understand the primary selection. If you are cutting and pasting between release-2 and release-3 X applications, the primary selection and the cut buffer can get out of step; and so when you paste, you may not get the text that you expect. To get around this problem, the COHERENT X release includes the X utility **xcutsel**, which copies text between the primary selection and the cut buffer. When you invoke this application, it displays a window that shows the following buttons:

`(quit)` Exit from **xcutsel**. **xcutsel** automatically releases all selections that it holds.

`(copy_PRIMARY_to_0)`
Copy the primary selection into the cut buffer.

`(copy_0_to_PRIMARY)`
Copy the cut buffer into the primary selection.

The button labels reflect the selection and cut buffer selected by command line options or through the resource database. By default, **xcutsel** uses the selection named **PRIMARY** and the cut buffer **CUT_BUFFER0**. You can override either or both of these by setting command-line arguments or resources.

When you click the button `(copy_0_to_PRIMARY)`, the button remains in reverse video as long as **cutsel** owns the selection, as a reminder. The value of the selection remains constant: if the cut buffer is changed, you must again activate the copy button to retrieve the new value.

Options

xcutsel recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-cutbuffer *number*
Set the cut buffer to use. The default is cut buffer 0.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-selection *name*
Set the name of the selection to use. The default is **PRIMARY**.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

X Defaults

This program accepts all of the standard X Toolkit resource names and classes as well as:

selection (class **Selection)**
Name the selection to use. The default is **PRIMARY**.

cutBuffer (class **CutBuffer)**
Set the number of the cut buffer to use. The default is Zero.

Widget Names

Use the following instance names when you wish to configure their labels:

sel-cut (class **Command)**
This is the button `(copy_SELECTION_to_BUFFER)`.

cut-sel (class **Command)**
This is the button `(copy_BUFFER_to_SELECTION)`.

quit (class **Command)**
This is the button `(quit)`.

See Also

xclipboard, xterm, X utilities

LEXICON

Notes

All X applications shipped with the COHERENT release of X conform to X release 3 or later. Therefore, you should never need to use this utility. You may need it, however, if you import antique X applications.

xcutsel has no way by which you can change the name of the selection or the number of the cut buffer while it is running.

xcutsel was written by Ralph R. Swick of DEC/MIT Project Athena.

xdpyinfo — X Utility

Display information about an X server
xdpyinfo [-display *displayname***]**

xdpyinfo displays information about the X server. You can use it to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the different types of screens and visuals that are available.

Environment

xdpyinfo reads the environmental variable **DISPLAY** to find the default host, display number, and screen.

See Also

xprop, xrdb, X utilities, xwininfo

Notes

xdpyinfo was written by Jim Fulton of the MIT X Consortium.

xedit — X Client

Simple text editor for X
xedit [-*toolkitoption ...* **] [***filename***]**

xedit is a simple text editor that works directly under X. When you invoke **xedit**, it displays a window that contains the following four areas:

Commands
: A set of buttons that allow you to exit **xedit**, save the file, or load a new file into the edit window.

Messages Messages from **xedit**. In addition, you can use this window as a scratch pad.

Filename The name of the file being edited, and whether it is **Read-Write** or **Read Only**.

Edit The text of the file that you are editing or creating.

Options

filename names the file to edit. If you do not name a file, **xedit** lets you load or create file after it has started up.

xedit recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
: Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
: Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Editing

xedit uses the Athena Text widget for the three sections that allow text input. The characters typed go into the Text widget that the mouse cursor is over. If the mouse cursor is not over a text widget, then keystrokes do nothing. This is also true for the special key sequences that pop up dialogue widgets, so typing **<ctrl-S>** in the filename widget enables searching in that widget, not the edit widget.

Both the message window and the edit window create a scrollbar if the mass of text to display is too large to fit within that window. Horizontal scrolling is not allowed by default, but can be turned on through the Text widget's resources.

xedit's default keystrokes closely resemble those of the MicroEMACS editor. The default keystrokes are as follows:

<ctrl-A> Move the cursor to the beginning of the current line.
<ctrl-B> Move the cursor one character to the left.
<ctrl-D> Delete the next character.
<ctrl-E> Move the cursor to the end of the line.
<ctrl-F> Move the cursor forward by one character.
<ctrl-G> Multiply reset.
<ctrl-H> Delete one character to the left.
<ctrl-J> Newline and indent.
<ctrl-K> Kill text from the cursor to the end of the line.
<ctrl-L> Redraw the display.
<ctrl-M> Newline.
<ctrl-N> Move the cursor to the next line.
<ctrl-O> Newline and backup.
<ctrl-P> Move the cursor to the previous line.
<ctrl-R> Search and replace backwards.
<ctrl-S> Search and replace forward.
<ctrl-T> Transpose two characters.
<ctrl-U> Multiply an argument by four.
<ctrl-V> Display the next pageful of text.
<ctrl-W> Kill selection.
<ctrl-Y> Yank text — copy back text that had been killed.
<ctrl-Z> Scroll one line up.
<esc>B Move the cursor one word to the left.
<esc>F Move the cursor one word to the right.
<esc>I Insert a file.
<esc>K Kill to end of the current paragraph.
<esc>Q Form a paragraph.
<esc>V Display the previous page of text.
<esc>Y Insert the current selection (i.e., text you have cut with the mouse).
<esc>Z Scroll one line down.
<esc>D Delete the next word to the right of the cursor.
<esc>D Kill the next word to the right of the cursor.
<esc>H Delete the word to the left of the cursor.
<esc>H Kill the word to the left of the cursor.
<esc>< Move the cursor to the beginning of the file.
<esc>> Move the cursor to the end of the file.
<esc>] Move the cursor forward by one file.
<esc>[Move the cursor to the beginning of the previous paragraph.
<esc><del> Delete one word to the left of the cursor.
<esc><shift><del>
Kill one word to the left of the cursor.
<esc><backspace>
Delete one word to the left of the cursor.
<esc><shift><backspace>
Kill one word to the left.

In addition, you can use the X system's default cut-and-paste feature, as follows:

Left Button Down	Start selection
Left Button Motion	Sweep out selection
Left Button Up	End selection (i.e., cut)
Middle Button Down	Insert current selection (paste)
Right Button Down	Extend current selection
Right Button Motion	Adjust selection
Right Button Up	End selection (cut)

If your mouse has only two buttons, press both buttons to mimic the middle mouse button. Note that this only works if you have un-commented the line **emulate3buttons** in the file **/usr/X11/lib/Xconfig**.

Commands

xedit's commands window has the following buttons:

`(Quit)` Quit the current editing session. If you have modified the file since you last saved it, **xedit** displays a warning message and gives you a chance to save it.

`(Save)` If file backups are enabled (see **Resources**, below), **xedit** stores a copy of the original, unedited file in **<prefix>***file***<suffix>**, then overwrites the *file* with the contents of the edit window. The file name is retrieved from the Text widget directly to the right of the **Load** button.

`(Load)` Load the file named in the text widget immediately to the right of the this button and display it in the **Edit** window. If the currently displayed file has been modified, a warning message asks you to save the changes or press **Load** again.

Resources

xedit uses the following resources:

enableBackups (Class **EnableBackups)**
Ask **xedit** to save the original version of a file into **<prefix>***file***<suffix>** before it saves changes to the file being edited. The default value for this resource is **off**, stating that no backups should be created.

backupNamePrefix (Class **BackupNamePrefix)**
Name the string that **xedit** prefixes to the name of the backup file. The default is **.BAK**.

Environment

xedit reads the following environmental variables:

DISPLAY
The default host and display number.

XENVIRONMENT
The name of a resource file that overrides the global resources stored in the property **RESOURCE_MANAGER**.

Files

/usr/X11/lib/app-defaults/Xedit — Required resources

See Also

X clients, xrdb

Notes

xedit was written by Chris Peterson of the MIT X Consortium.

xev — X Utility

Print contents of X events

xev [-display *displayname***] [-geometry** *geometry***] [-bw** *pixels***]**
[-bs (NotUseful|WhenMapped|Always)][-id *windowid***] [-s] [-name** *string***] [-rv]**

xev creates a window and then asks the X server to send it *events* whenever anything happens to the window (such as being moved, resized, typed in, clicked in, etc.). You can also attach **xev** to an existing window. It is useful for seeing what causes events to occur and to display the information that they contain.

Options

xev recognizes the following command-line options:

-bs (NotUseful|WhenMapped|Always)
Specify the kind of backing storage to give the window. The default is **NotUseful**.

-bw *pixels*
Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Contact *host* on *server*.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-id *windowid*
Monitor window *windowid*, rather than creating a new window.

-name *string*
Assign *name* to the newly created window.

-rv Display the window in reverse video.

-s Enable save-unders on the window.

See Also

xwininfo, xdpyinfo, X utilities

Notes

xev was written by Jim Fulton of the MIT X Consortium.

xeyes — X Client

Display two roving eyes

xeyes

The X client **xeyes** displays two oval "eyes" on the screen. The pupil of each "eye" rotates in its "eyeball" to follow the mouse pointer around the screen. This can help you find the mouse cursor in a crowded screen.

xeyes recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the circle around each eye to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*. **xeyes** restricts the foreground color to the "pupil" of each "eye". Thus, the command

```
xeyes -bd red -fg red
```

displays a pair of bloodshot eyes.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width×height±xoffset±yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

See Also

X clients

xfd — X Utility

Display all the characters in an X font
xfd [-toolkotoptions ...] -fn *fontname*

The utility **xfd** opens a window that displays the name of the font being displayed, a row of command buttons, several lines of text for displaying character metrics, and a grid that contains one glyph per cell. The characters are shown in increasing ASCII order from left to right, top to bottom. The first character displayed at the top left is character zero unless the you have used the command-line option **-start** (described below), in which case **xfd** uses the character with the number given in the option **-start**.

xfd displays the characters in a grid of boxes, each large enough to hold any character in the font. Each character glyph is drawn using the PolyText16 request (used by the **Xlib** routine **XDrawString16**).

If you use the option **-box**, **xfd** draws a rectangle around each character that shows where an ImageText16 request (used by the **Xlib** routine **XDrawImageString16**) would display background color.

The origin of each glyph is normally set so that the character is drawn in the upper left hand corner of the grid cell. However, if a glyph has a negative left bearing or an unusually large ascender, descender, or right bearing (as is the case with the **cursor** font), some characters may not appear in their own cells. Use the command-line option **-center** to force **xfd** to center each glyph within its cell.

xfd displays the following buttons in its window:

`(Next_Page)`
xfd may not be able to fit all of a font's characters into the window at once. Press this button to see the next page of glyphs.

`(Prev_Page)`
Display the previous page of glyphs.

`(Quit)` Exit **xfd**.

To display the metrics (index, width, bearings, ascent, and descent) for a character, click its cell in the display area.

The font name displayed at the top of the window is the full name of the font, as determined by the server. The command **xlsfonts** generates lists of fonts, as well as more detailed summaries of their metrics and properties.

Options

xfd recognizes the following command-line options:

-bc *color* Use *color* if ImageText boxes are drawn.

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-box Draw a box around each character to outline the area that a request to ImageText would fill with background color.

-bw *pixels* Set the width of the border to *pixels*.

-center Center each character within its cell.

-display *host*[**:***server*][**.***screen*]
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Display *font*.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-start *number*
Display the font beginning with character *number*. The default is zero.

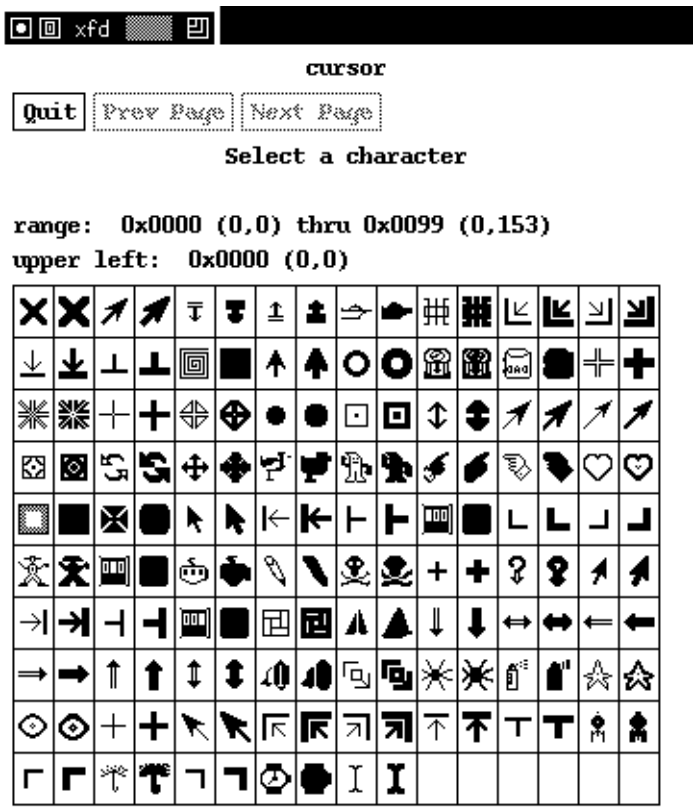
-xrm *resourcestring*
Use *resourcestring* to define a resource.

Example

To display the cursor font — that is, the font of predefined shapes for the mouse cursor — type the command:

```
xfd -center -fn cursor
```

xfd displays the following window on your screen:



Note that a cursor consists of two bit-mapped images: the cursor itself, and a mask that goes around it. The names of the cursors are defined in file **/usr/X11/include/X11/cursorfont.h**.

See Also

xfontsel, xlsfonts, xrdb, X utilities

LEXICON

Notes

xfd does not skip pages full of non-existent characters.

xfd was written by Jim Fulton of MIT X Consortium, based on a previous program of the same name that was written by Mark Lillibridge of MIT Project Athena.

xfontsel — X Utility

Interactively select X11 fonts
xfontsel [-*toolkitoption ...***] [-pattern** *fontname***] [-print] [-sample** *text***] [-sample16** *text16***] [-noscaled]**

xfontsel provides a simple way to display the fonts known to your X server, examine samples of each, and retrieve the full X Logical Font Description (XLFD) for a font.

Options

xfontsel recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-noscaled Disable the ability of **xfontsel** to select scaled fonts at arbitrary pixel or point sizes. This makes it clear which bitmap sizes are advertised by the server, and can avoid an accidental and sometimes prolonged wait for a font to be scaled.

-pattern *font*
Examine *font*. If you do not use this option, you can select interactively from among all all fonts with XLFD 14-part names. To work with only a subset of the fonts, specify **-pattern** followed by a partially or fully qualified font name; e.g.,

```
-pattern \*medium\*
```

selects a font only if its name contains the string. Be careful to escape wildcard characters so the COHERENT shell does not interpret them.

-print Write the selected font specifier to standard output when you click the `(quit)` button. Regardless of whether you have specified **-print**, you can make the font specifier the primary (text) selection by clicking the `(select)`button.

-rv Simulate reverse video by exchanging the foreground and background colors.

-sample *text*
Set the sample text to be used to display the selected font if the font is linearly indexed.

-sample16 *text16*
Set the sample text to be used to display the selected font if the font is matrix encoded.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Interactions

When you invoke **xfontsel**, it draws the following window on your screen:

As you can see, the window is divided into four areas:

- The top area displays (quit) and (select) buttons, and a message about the number of fonts that match the description you entered.
- The second area displays 14 buttons, one for each field in a font's XLFD name. Clicking on one of these buttons presents the possibile entries for the corresponding XLFD field. When you select an item from a menu, **xfontsel** limits the items available in all other menus to those which correlate with what you have already selected; to make other values selectable, you must first de-select some other field or fields by choosing the '*' entry in its menu. To omit some values altogether, set the resource **ShowUnselectable**, as described below. These buttons are described below.
- The third field displays the XLFD name. Unless you have narrowed the selection of fonts, each of the 14 fields is represented by an asterisk, separated by an underscore.
- The bottom displays a font, as constructed from the XLFD entered in the above two fields.

The following describes the 14 buttons in the second field of the screen. These buttons define, in order, the 14 fields in a font's XLFD name, as follows:

(fndry) Foundry: the company that made the fonts, e.g., Bitstream or Adobe.

(fmly) Family: the family of typefaces, e.g., **courier** or **gothic**.

(wght) Weight of the typeface, e.g., **bold** or **medium**.

(slant) The angularity of the font. **nil** indicates nothing — that is, no typeface; **i**, italic; **o**, slanted Roman; and **r**, Roman.

(sWdth) Set width of the face, that is, how closely the characters are set next to each other. **normal** indicates normal width; **semicondensed** sets the text somewhat more tightly.

(adstyl)
The style of text: **sans** indicates a sans-serif face.

(pxlsz) Pixel size: the size of the face in pixels.

(ptSz) Point size: the size of the font in decipoints — that is, tenths of a printer's point. There are 720 decipoints per inch.

(resx) Horizontal (X) resolution, in pixels per inch.

(resy) Vertical (Y) resolution, in pixels per inch.

(spc) Spacing; for example, **m** indicates monospace.

(avgWdth)
Average width, in tenths of a pixel.

(rgstry)
Registry, that is, the organization that registered the character set embodied in this font.

(encdng)
Encoding: the name of character set embodied in this font. Note that the commonest character set is ISO-8859-1, also called ISO Latin 1, which describes the ASCII character set normally used in the United States.

Whenever you change the value of an XLFD field, **xfontsel** assert ownership of the **primary_font** property. Other applications (see, e.g., **xterm**) can then retrieve the specified font.

Scalable fonts come back from the server with zero for the pixel size, point size, and average width fields. Selecting a font name with a zero in these positions results in an implementation-dependent size. Any pixel or point size can be selected to scale the font to a particular size. Any average width can be selected to scale the font anamorphically (although you may find this challenging given the size of the average width menu).

Clicking the left pointer button in the (select) button causes the currently selected font name to become the primary text selection, as well as the **primary_font** selection. This allows you to paste the string into other applications. The (select) button remains highlighted to remind you of this fact; it de-highlights when some other application seizes the primary selection. The (select) button is a toggle: pressing it when it is highlighted causes **xfontsel** to release ownership of the selection and de-highlight the button. Clicking the (select) widget twice is the only way to force **xfontsel** to release the **primary_font** selection.

LEXICON

Resources

The application's class is **XFontSel**. Most of its user interface is configured in the configuration file **/usr/X11/lib/app-defaults/XFontSel**. If this file is missing, **xfontsel** prints a warning message to standard output, and the resulting window will be nearly incomprehensible.

The following gives application-specific resources:

cursor (class **Cursor)**
Set the cursor for the application window.

pattern (class **Pattern)**
Set the font-name pattern to select a subset of available fonts. This is equivalent to the command-line option **-pattern**. Most useful patterns contain at least one field delimiter; e.g., "*-m-*" for monospaced fonts.

pixelSizeList (class **PixelSizeList)**
Give a list of pixel sizes to add to the pixel-size menu, so that scalable fonts can be selected at those pixel sizes. The default **pixelSizeList** contains 7, 30, 40, 50, and 60.

pointSizeList (class **PointSizeList)**
Give a list of point sizes (in units of tenths of points) to add to the point size menu, so that scalable fonts can be selected at those point sizes. The default **pointSizeList** contains 250, 300, 350, and 400.

printOnQuit (class **PrintOnQuit)**
If set to **True**, the currently selected font name is printed to standard output when you click the `(quit)` button. Equivalent to the **-print** option.

sampleText (class **Text)**
The sample one-byte text to use for linearly indexed fonts. Each glyph index is a single byte, with a newline character separating lines.

sampleText16 (class **Text16)**
The sample two-byte text to use for matrix-encoded fonts. Each glyph index is two bytes, with a newline character separating lines.

scaledFonts (class **ScaledFonts)**
If set to **True**, enable selection of arbitrary pixel and point sizes for scalable fonts.

The following gives the widget resources used by **xfontsel**:

showUnselectable (class **ShowUnselectable)**
For each field menu, indicate whether to show values that are not currently selectable, based upon previous field selections. If shown, the unselectable values are clearly identified as such and do not highlight when the pointer is moved down the menu. The full name of this resource is

```
fieldN.menu.options.showUnselectable
```

of class:

```
MenuButton.SimpleMenu.Options.ShowUnselectable
```

where *N* is replaced with the field number (starting with the left-most field, which is numbered zero). The default is **True** for all but field 11 (average width of characters in font) and **False** for field 11. If you never want to see unselectable entries,

```
*menu.options.showUnselectable:False
```

is a reasonable entry for your personal resource file.

Files

/usr/X11/lib/app-defaults/XFontSel— Resource file

See Also

xfd, xrdb, X utilities

Notes

Sufficiently ambiguous patterns can be misinterpreted and lead to an initial selection string that may not correspond to what you intended, and that may cause the initial sample text output to fail to match the proffered

string. Selecting any new field value corrects the sample output, though possibly resulting in no matching font.

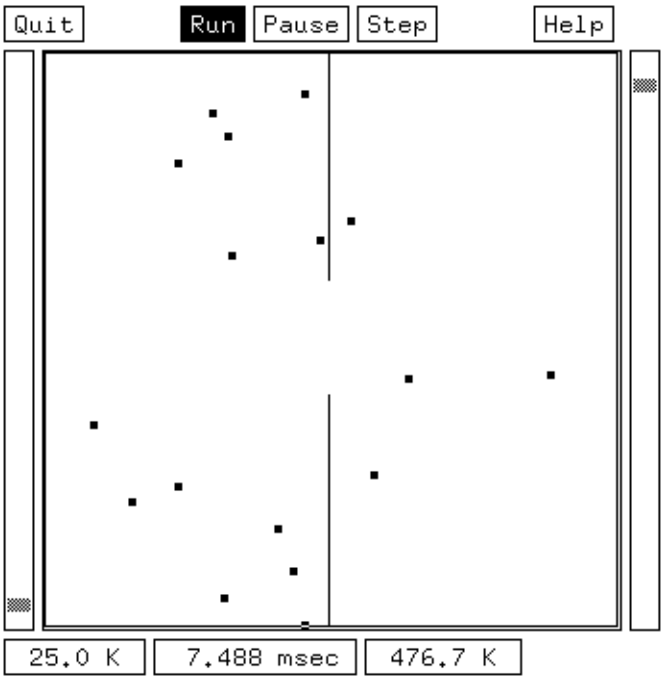
When running on a slow machine, you may request a field menu before the font names have been completely parsed. **xfontsel** prints to the standard error an error message that indicates a missing menu, but otherwise nothing happens.

xfontsel was written by Ralph R. Swick of Digital Equipment Corporation/MIT Project Athena.

xgas — X Client

Animated simulation of an ideal gas
xgas [-*option ...*]

xgas models an ideal gas in a heated chamber. The chamber is partially divided into two parts. You can control the temperature of each part independently by clicking on a scrollbar:



Each dot represents a molecule. The "molecules" move within the box; when they touch a wall, they assume the temperature to which you have set that part of the chamber. Their velocities depend upon their temperature.

Click the left mouse button to create a molecule at the cursor's position. Click the right mouse button to create the maximum number of molecules at the cursor's position.

The **xgas** window displays the following four buttons:

`(Quit)` Exit from **xgas**.

`(Run)` Run the game continually.

`(Pause)`Stop running the game. To resume running, click `(Run)` again.

`(Step)` Run the game one step at a time. This lets you examine in detail how the "molecules" move and are influenced by the temperatures of the chamber.

`(Help)` This option displays a help message on how to run **xgas**. If you wish, you can also read a document on the physics of an ideal gas and how **xgas** models them.

Options

xgas recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host*[**:***server*][**.***screen*]
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

X Defaults

xgas uses the following X resources:

timeStepSize
Set the simulated time duration, in microseconds, for each cycle of computation.

delay Set the real time interval between timestep computations.

randomBounce
Each time a molecule collides with a wall, it bounces elastically (angle of incidence equals angle of reflection). This default adds a component of randomness to this angle. **RandomBounce** varies from 0.0 (no randomness) to 1.0 (completely random angle of incidence).

equilibrium
Each time a molecule collides with a wall, its kinetic energy approaches that of the temperature of the wall. If **equilibrium** is 1.0, the molecule reaches the wall temperature immediately. For values between 1.0 and 0.0, the molecule approaches the temperature of the wall more slowly.

maxMolecules
Set the maximum number of molecules.

Files

/usr/X11/lib/app-defaults/XGas — Default resource file

See Also

ico, X clients, xwd

Notes

When the chamber is resized, molecules should be rearranged appropriately. Instead, the molecule arrays are reinitialized.

xgas was written by Larry Medwin.

xgc — X Client

X graphics demonstration
xgc [-*toolkitoption ...***]**

xgc demonstrates features of the X graphics primitives. By manipulating the mouse, clicking buttons, pressing mouse buttons, and otherwise playing with the window, you can learn how these objects work.

Options

xgc recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host*[**:***server*][**.***screen*]
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Environment

xgc reads the following environmental variables:

DISPLAY
Find the default host and display number.

XENVIRONMENT
Find the name of a resource file that overrides the global resources stored in the **RESOURCE_MANAGER** property.

See Also

X clients

Notes

xgc was written by Dan Schmidt of MIT.

xinit — X Utility

Initiate the X Window System
xinit [[*client***] *options***] [--[***server***] [***display***] *options***]**

xinit launches the X Window System's server and a first client program. When this first client exits, **xinit** terminates the X server and closes the X Windows session.

xinit Scripts

xinit executes the contents of a script, which invokes various X clients. By default, **xinit** executes the contents of script **$HOME/.xinitrc**. If this file doesnot exit, **xinit** executes the script **/usr/X11/lib/xinit/xinitrc**. The tutorial *X Windows Clients*, at the beginning of this manual, gives examples of how to create your own **.xinitrc** file and modify it to suit your tastes.

The programs that are invoked within **$HOME/.xinitrc** should be run in the background if they do not exit immediately, so that they do not prevent other programs from starting up. However, the last long-lived program that **$HOME/.xinitrc** launches (usually the window manager **twm**) should be left in the foreground so that the script will not exit (which tells **xinit** that the user is done and it should exit). For example, the default file **xinitrc** contains the following commands:

```
xsetroot -bitmap /usr/X11/include/X11/bitmaps/wide_weave
xclock -geometry 135x141+15+26 -fg blue -chime -update 1 &
xterm -ls -geometry 80x24+130+146 -cr red &
twm
```

The command **xsetroot** tiles the background of the screen with a bit-mapped image. Commands **xclock** and **xterm** invoke those clients and display them on the screen; note that both are run in the background. Finally, the command **twm** invokes the window manager **twm** in the foreground. When **twm** exits, **xinit** shuts down the X server and returns you to the normal character-based COHERENT interface.

Command-line Options

xinit itself normally is invoked through the script **startx**. You can modify this script, if you wish, to set **xinit**'s options. You can use command-line arguments to have **xinit** ignore the contents of its **xinitrc** files.

You can name an alternate client or server on **xinit**'s command line. Give the desired client and its arguments as the first command-line arguments to **xinit**. To specify a command line for a server, append two hyphens '--' to **xinit**'s command line (after any client and arguments), followed by the commands you wish to pass to the server.

The name of a client program and the name of a server program must each begin with a slash '/' or a period. If they do not, **xinit** treats them as arguments to be appended to their respective startup lines. This lets you add arguments (for example, foreground and background colors) without having to retype the whole command line.

If **xinit**'s command line does not name a server, and the first argument following the pair of hyphens '--' is a colon followed by a digit, **xinit** uses that number to identify the display. (X Windows for COHERENT at present supports only the default display, which is display number 0.) **xinit** appends all remaining arguments onto the server's command line. For a list of the options you can pass to the X server, see the Lexicon entry for **X**.

The following gives examples of **xinit**'s command-line arguments. The first example

```
xinit -- /usr/bin/X11/Xqdss  :1
```

launches the X server named **Xqdss** on display number one.

The next example

```
xinit -geometry=80x65+10+10 -fn 8x13 -j -fg white -bg navy
```

launches the server named **X**, and invokes **xterm** with arguments to set its geometry, font, foreground and background colors, and with jump scrolling turned on. It ignores **$HOME/.xinitrc**.

The command

```
xinit -e widgets -- ./Xsun -l -c
```

uses the command **./Xsun -l -c** to launch the server. It also launches **xterm** by default, and passes it the argument **-e widgets**.

Finally the command

```
xinit /bin/rsh fasthost cpupig -display ws:1 --  :1 -a 2 -t 5
```

launches the server named **X** on display 1 with the arguments **-a 2 -t 5**. It then starts a remote shell to run the command **cpupig** on the machine **fasthost**, telling it to display back on the local workstation.

Environment Variables

xinit reads the following environmental variables:

DISPLAY
The name of the display to which clients should connect.

XINITRC
Name the initialization file to execute.

Files

$HOME/.xinitrc — Client script
/usr/X11/lib/xinit/xinitrc — Default client script
xterm — Client to run if no client script exists
$HOME/.xserverrc — Default server script
X — Server to run if **.xserverrc** does not exist

See Also

startx, X, xterm, X utilities

Notes

xinit was written by Bob Scheifler of the MIT Laboratory for Computer Science.

xkill — X Utility

Kill an X client
xkill [*options***]**

xkill commands the X server to close its connection to a client. The severed client dies and its window vanishes from the screen. The killed client does not clean up after itself, so "stuff" may be left littering memory or the file system. If used carelessly, this program can damage your system, but it is useful for aborting programs that insist on displaying undesired windows on your screen.

If you do not indicate the client to kill by using the **-id** command-line option (described below), **xkill** displays a skull-and-crossbones cursor and kills the client whose window you click with it.

xkill recognizes the following command-line options:

-all Kill all clients with top-level windows on the screen. **xkill** asks you to confirm the killing by pressing one of the mouse buttons. Do not use this option unless you absolutely must.

-button [*number* **any]**
When *button* is pressed, kill the X client whose window is being clicked. **any** indicates that any button will do. By default, the leftmost button does the trick.

-display *host***[:***server***][.***screen***]**
Specify a resource on *screen* of *server* on host system *host*.

-frame Ignore the standard conventions for finding top-level client windows (which, typically, are nested inside a window-manager window), and instead simply believe that you want to kill direct children of the root.

-id *resource*
Kill the client whose resource has the identifier *resource*.

Resources

xkill uses the resource **Button**, which specifies the mouse button to press when selecting a window for death. **any** indicates any button.

See Also

X utilities

Notes

xkill was written by Jim Fulton and Dana Chee.

xload — X Client

Display your system's load average
xload [-*toolkitoption ...***] [-scale** *integer***] [-update** *seconds***] [-hl** *color***] [-highlight** *color***]**
[-jumpscroll *pixels***] [-label** *string***] [-nolabel] [-lights]**

xload opens a window and displays a histogram of your system's load average. The histogram is a bar graph that grows from left to right.

xload recognizes the following command-line options:

-bd *color* Set the color of the border to *color*. The default is black

-bg *color* Set the color of the background to *color*. The default is white.

-bw *pixels* Set the width of the border to *pixels*. The default is two.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*. The default is black.

-fn *font* Use *font* in the display. The default is **fixed**, a 6×10, fixed-width font.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-highlight *color*
Specify the color of the scale lines.

-jumpscroll *pixels*
Jump the histogram by *pixels* when it reaches the edge of the window. The default is half the width of the current window. To obtain smooth scrolling, set *pixels* to one.

-label *string*
Display *string* as a label above the histogram.

-nolabel Display no label above the histogram.

-rv Simulate reverse video by exchanging the foreground and background colors.

-scale *ticks* Specify the minimum number of tick-marks in the histogram, where one tick-mark represents one load-average point. If the load goes above this number, **xload** creates more divisions, but it will never use fewer than *ticks*. The default is one.

-update *seconds*
Update the histogram after an interval of *seconds*. The default is five. If the histogram is iconified and then de-iconified, **xload** updates it immediately.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Resources

xload uses the resource **showLabel(**class **Boolean)**. If false, **xload** displays no label.

Environment

xload reads the following environmental variables:

DISPLAY
The default host and display.

XENVIRONMENT
The name of a resource file that overrides the global resources stored in the manager **RESOURCE_MANAGER**.

See Also

X clients
COHERENT Lexicon: **ls**

Notes

xload must be able to open and read file **/dev/kmem**. Sites that do not allow general access to this file should have **xload** belong to the same group as **/dev/kmem**, and turn on the **set group id** flag. For details on file permissions, see the entry for the command **ls** in the COHERENT Lexicon.

xload was written by K. Shane Hartmann and Stuart A. Malone. Jim Gettys, Bob Scheifler, Tony Della Fera, and Chris Peterson added features.

xlogo — X Client

Display the X Window System logo
xlogo [*option ...***]**

xlogo displays the X Window System logo in its own window. It is simply a wrapper around the Athena Logo widget. It recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-rv Simulate reverse video by exchanging the foreground and background colors.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Resources

xlogo uses the Logo widget in the Athena widget set. It understands all of the core resource names and classes, as well as the following:

foreground(class **Foreground)**
Give the foreground color of the logo. If **reverseVideo** is specified, the default color is white; otherwise, it is black.

height(class **Height)**
Give the height of the logo.

reverseVideo(class **ReverseVideo)**
Indicate reverse video by exchanging the foreground and background colors.

width(class **Width)**
Give the width of the logo.

Environment

xlogo reads the following environmental variables:

DISPLAY
The default host and display.

XENVIRONMENT
The name of a resource file that overrides the global resources stored in the property **RESOURCE_MANAGER**.

Files

/usr/X11/lib/app-defaults/XLogo — Resource file

See Also

X clients

Notes

The Athena Logo widget is undocumented.

xlogo was written by Ollie Jones and Jim Fulton of the MIT X Consortium, based on a graphic design by Danny Chong and Ross Chapman of Apollo Computer.

xlsatoms — X Utility

List atoms defined on server
xlsatoms [-display *dpy***] [-format** *string***] [-range [***low***] [-***high***]] [-name** *string***]**

xlsatoms lists the atoms being used by the X server. By default, it lists all atoms starting from one (the lowest atom value defined by the protocol) until it finds an atom that is undefined. If an explicit range is given, **xlsatoms** tries all atoms in the range, regardless of whether any are undefined.

LEXICON

Options

xlsatoms recognizes the following command-line options:

-display *dpy*
Connect to display number *dpy*.

-format *string*
Format the output. *string* is a **printf()**-style format string that **xlsatoms** uses to format an atom. An atom consists of the arguments *value* and *name*, in that order; the former is an **unsigned long** and the latter a **char** *. **xlsatoms** automatically supplies a newline at the end of each line. The default formatting string is **%ld\t%s**.

-name *string*
Name an atom to list. If the atom *string* does not exist, **xlsatoms** prints a message on the standard error device.

-range [*low***]-[***high***]**
Set the range of atoms to check, by value. If *low* is not given, **xlsatoms** assumes a value of one. If *high* is not given, **xlsatoms** stops at the first undefined atom at or above *low*.

Environment

xlsatoms reads the environmental variable **DISPLAY** to find the host and display to use.

See Also

xprop, X utilities

Notes

xlsatoms was written by Jim Fulton of the MIT X Consortium.

xlsclients — X Utility

List client applications running on a display
xlsclients [-display *displayname***] [-a] [-l] [-m** *maxcmdlen***]**

The X utility **xlsclients** lists information about the client applications running on a display. You can use it to generate a script that represents a snapshot of your current session.

xlsclients recognizes the following command-line options:

-a List the clients on all screens. By default, **xlsclients** lists only the clients on the default screen.

-display *displayname*
List information about the display *displayname*.

-l By default, **xlsclients** prints the machine name and command string. This options tells **xlsclients** to output a long listing that also gives the window name, the icon name, and class hints.

-m *maxcmdlen*
Give the maximum number of characters to print for a given command. The default is 10,000.

Environment

xlsclients reads the environmental variable **DISPLAY** to get the host, display number, and screen.

See Also

X utilities, xwininfo

Notes

xlsclients was written by Jim Fulton of the MIT X Consortium.

xlsfonts — X Utility

List fonts being used on a server
xlsfonts [-*option ...***] [-fn** *pattern***]**

xlsfonts lists the fonts that match the given *pattern*. *pattern* can contain the wildcard characters '*' and '?', which match, respectively, any sequence of characters (including none) and any single character. If its command line gives no pattern, **xlsfonts** uses '*' by default. Note that you must the '*' and '?' to protect them from being expanded by the shell.

Options

xlsfonts recognizes the following command-line options:

-1 Listings should use a single column. This is the same as **-n 1**. (Note that the character displayed here is the numeral one, not a lower-case el.)

-display *host***:***dpy*
Contact display *dpy* on X server *host*.

-l[l[l]] Set the length of the listing for each font. The options respectively request medium-long, long, and very long listings. (Note that the character displayed here is a lower-case el, not the numeral one.)

-m Long listings should also print the minimum and maximum bounds of each font.

-C Listings should use multiple columns. This is the same as **-n 0**.

-o Tell **xlsfonts** to perform an **OpenFont** (and **QueryFont**, if appropriate) rather than a **ListFonts**. This is useful if **ListFonts** or **ListFontsWithInfo** fail to list a known font (as is the case with some scaled-font systems).

-n *columns*
Use *columns* columns in the output. By default, **xlsfonts** fits as many columns of font names as it can into the number of characters specified by the option **-w**.

-w *width*
Give the width of the printout, in characters. **xlsfonts** uses this value to compute the number of columns to print. The default is 79.

-u Leave the output unsorted.

Environment

xlsfonts reads the environmental variable **DISPLAY** to find the host and display to use.

See Also

xfd, xset, X utilities

Notes

The command **xlsfonts -lll** can tie up your X server for a very long time. *Caveat utilitor.*

xlsfonts was written by Mark Lillibridge of MIT Project Athena, Jim Fulton of the MIT X Consortium, and Phil Karlton of SGI.

xmag — X Client

Magnify a part of the screen
xmag [-mag *factor***] [-source** *geom***] [-***toolkitoption ...* **]**

The X client **xmag** magnifies a portion of an X screen. If its command line does not explicitly specify a region of the screen to magnify, **xmag** lets you select interactively the portion of the screen to magnify. It displays a square with the mouse cursor in its upper-left corner; to select a portion of the screen, drag the square to the area that interests you and click the left mouse button.

When you have selected a region to magnify, **xmag** pops up a window shows a magnified version of the selected region. The following example shows the **xmag** window magnifying part of itself:

As you can see, the magnification represents each pixel in the source image by a small square; on a color screen, the square is the same color of the original pixel.

At the top of the window, **xmag** displays the following six buttons:

(close) Close the **xmag** window.

(replace)
Again display the selector square, so you can select a different portion of the screen to magnify. The newly selected portion replaces the portion displayed in the current **xmag** window. You can select a portion of the **xmag** window itself, which in effect lets you "zoom in" on a portion of the screen.

(new) Again display the selector square, so you can select another portion of the screen to magnify. **xmag** displays the newly selected portion in another **xmag** window.

(select)
Cut the magnified image into the primary selection.

(paste) Copy into the primary selection the display area of the current **xmag** window. Note that you can cut and paste between **xmag** and the X client **bitmap**.

Resizing the **xmag** window resizes the magnification area. **xmag** preserves the color map, visual depth, and window depth of the source image being magnified.

Options

xmag recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-bw *pixels* Set the width of the border to *pixels*.

-display *host***[:***server***][.***screen***]**
Display the client's window on screen number *screen* of *server* on host system *host*.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-mag *integer*
Use *integer* as the magnification factor. The default is five.

-rv Simulate reverse video by exchanging the foreground and background colors.

-source *geom*
Set the size and location of the source region on the screen. By default, **xmag** provides a 64×64-pixel square with which you can select an area of the screen.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

See Also

bitmap, X clients

Notes

xmag was written by Dave Sternlicht and Davor Matic of the MIT X Consortium.

xmkmf — X Utility

Control the building of a Makefile
xmkmf [-a] [*topdir* **[***curdir***]]**

The X utility **xmkmf** is a script that invokes the utility **imake** to build a **Makefile** from an **Imakefile**.

When invoked with no arguments and in a directory that contains an **Imakefile**, **xmkmf** reads the **Imakefile** and the configuration files designed for X Windows for COHERENT, and generates a **Makefile**.

When invoked with the option **-a**, **xmkmf** builds the **Makefile** in the current directory, then automatically executes the commands

```
make Makefiles
```

(in case there are subdirectories),

```
make includes
```

and:

```
make depends
```

This is the normal way to configure software that is outside the MIT X build tree.

If you are working within the MIT X build tree specify the argument *topdir* as the relative path name from the current directory to the top of the build tree. You can also specify *curdir* as a relative path name from the top of the build tree to the current directory. You must supply *curdir* if the current directory has subdirectories, or if the *Makefile* cannot build the subdirectories. If you give **xmkfm** a *topdir*, it assumes that nothing is installed on your system, and looks for files in the build tree instead of using the installed versions. Note that it is unlikely that you will ever need to do this unless you are an X developer, and even then this option is seldom used.

For a complete description of what **xmkmf** does, read the script **/usr/X11/bin/xmkmf**.

See Also

imake, X utilities

Notes

Makefiles generated by **xmkmf** must be processed through **gmake**, not the default COHERENT implementation of **make**. If you attempt to use COHERENT **make** with such a **Makefile**, the compilation will fail with a number of mysterious errors.

xmodmap — X Utility

Modify X keymaps
xmodmap [-*option ...***] [***filename***]**

xmodmap edits and displays the keyboard's *modifier map* and *keymap table* that client applications use to convert event keycodes into keysyms. It works by executing one or more expressions that you either pass to it either through a script or enter from your keyboard. **xmodmap** usually is invoked from within a user's startup script, to

execute a script that configures the keyboard to his taste.

Options

xmodmap recognizes the following command-line options:

filename Name a file of **xmodmap** expressions to execute. This file is usually kept in a user's home directory and given a name like **.xmodmaprc**. A *filename* of **-** tells **xmodmap** to read the standard input.

-display *display*
Run **xmodmap** on *display*.

-e *expression*
Execute *expression*, as if it were read from a script. The command line can hold an indefinite number of **-e** options.

-grammar Print on the standard error a help message that describes the grammar of **xmodmap**'s expressions.

-help Print on the standard error a brief description of the command-line arguments. **xmodmap** does this whenever it encounters an option that it does not recognize on its command line.

-n Print the actions to perform, but do not actually change the mappings. Use this option to help debug a script.

-pk Print on the standard output the current keymap table.

-pke Print on the standard output the current keymap table in the form of expressions that can be fed back to **xmodmap**. Use this to prepare a template script that you can modify to suit your preferences.

-pm Print on the standard output the current modifier map.

-pp Print on the standard output the current mapping of the mouse (pointer) buttons.

-quiet Turn off the verbose logging. This is the default.

-verbose Print logging information as **xmodmap** parses its input.

Expression Grammar

xmodmap parses all of its expressions before it executes any of them. This lets it refer in a natural way to keysyms that are being redefined, without having to worry about name conflicts.

The following summarizes the expressions that **xmodmap** recognizes:

add *modifiername* **=** *keysymname ...*
Add to the indicated modifier map all keys contain that the given keysyms. **xmodmap** evaluates the keysym names after it has read all input expressions; this makes it easier for **xmodmap** to write expressions to swap keys.

clear *modifiername*
Remove all entries in the modifier map for *modifiername*, which must be one of the following strings: **Shift**, **Lock**, **Control**, **Mod1**, **Mod2**, **Mod3**, **Mod4**, or **Mod5**. Case does not matter in modifier names, although it does matter for all other names. For example, the expression

```
clear Lock
```

removes all keys that are bound to the shift-lock modifier.

keycode *number* **=** *keysymname ...*
Assign each *keysymname* to keycode *number*, which may be specified in decimal, hexadecimal, or octal. If you need a table of keycodes, run the program **xev**.

keysym *keysymname* **=** *keysymname ...*
Link each *keysymname* to the right of the equal sign to the keycode that is represented by the *keysymname* to the left of the equal sign. The list of keysym names are kept in files **/usr/X11/include/X11/keysymdef.h** and **/usr/X11/lib/XKeysymDB**. If a keysym is bound to multiple keys, the expression is executed for each matching keycode.

pointer = default
Remap the mouse (pointer) buttons to their default settings (i.e., button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = *code ...*
Map the mouse (pointer) buttons to the the indicated button codes. This expressions maps each *code* to the corresponding physical button, counting from left to right.

remove *modifiername* **=** *keysymname ...*
Remove every key that contains the given keysyms from the indicated modifier map. Unlike **add**, **xmodmap** evaluates the keysym names as it reads the line. This lets you remove keys from a modifier without having to worry about whether they have been reassigned.

xmodmap treats as a comment every line that begins with an exclamation point '!': If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

Examples

Many mice are designed so that the first button is leftmost, where it can be pressed by the index finger of the right hand. People who are left-handed frequently find it more comfortable to reverse the button codes, so that the first button is pressed by the index finger of the left hand. The first example reverses the button mapping on a three-button mouse:

```
xmodmap -e "pointer = 3 2 1"
```

Many editor applications support the notion of **Meta** keys. These resemble **Control** keys except that **<esc>** is held down instead of **<ctrl>**. However, some servers do not have a **Meta** keysym in their default keymap tables, so you must add one by hand. The following command attach **Meta** to the Multi-language key (sometimes labeled **Compose Character**). It also takes advantage of the fact that applications that need a **Meta** key simply need to get the keycode and do not require the keysym to be in the first column of the keymap table. This means that an application that is looking for a **Multi_key** (including the default modifier map) will not notice any change:

```
xmodmap -e "keysym Multi_key = Multi_key Meta_L"
```

One of the simpler, yet more convenient, uses of **xmodmap** is to set the keyboard's *rubout* key to generate an alternate keysym. This frequently involves exchanging **Backspace** with **Delete** to be more comfortable to the user. If the resource **ttyModes** in **xterm** is set as well, every terminal-emulator window will use the same key for erasing characters:

```
xmodmap -e "keysym BackSpace = Delete"
echo "XTerm*ttyModes:  erase ^?" | xrdb -merge
```

Some keyboards do not automatically generate the characters '<' and '>' characters when the (,) and (.) keys are shifted. The following script remedies this to reset the bindings for the comma and period:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
keysym period = period greater
```

One of the more irritating differences between keyboards is the location of the (ctrl) and (Caps_Lock) keys. The following script swaps them:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keysym Control_L = Caps_Lock
keysym Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

xmodmap's command **keycode** is useful for assigning the same keysym to multiple keycodes. Although unportable, it also permits you to write scripts that can reset the keyboard to a known state. The following script sets the (Backspace) key to generate **Del** (as shown above), flushes all existing bindings for the (Caps_Lock) key, makes the (Caps_Lock) key be a control key, makes (F5) generate **Escape**, and makes the (Break) key be a shift lock:

```
!
! On the HP, the following keycodes have key caps as listed:
!
!     101  Backspace
!      55  Caps
!      14  Ctrl
!      15  Break/Reset
!      86  Stop
!      89  F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

Example

For an example of how to use **xmodmap** to reconfigure a keyboard, see the file **/usr/X11/lib/.Xmodmap.ger**. This reconfigures the keyboard for an **xterm** or **xvt** session to follow the German keyboard layout.

Environment

xmodmap reads the environmental variable **DISPLAY** to find the number of the host and display.

See Also

xev, X utilities
COHERENT Lexicon: **keyboard tables, nkb**

Notes

Every time **xmodmap** evaluates a **keycode** expression, the server generates a **MappingNotify** event on every client. This can cause some thrashing. You should batch all changes and do them at once. Clients that receive keyboard input and ignore **MappingNotify** events will not notice any changes made to keyboard mappings.

xmodmap should generate **add** and **remove** expressions automatically whenever a keycode that is already bound to a modifier is changed.

xmodmap was written by Jim Fulton of MIT X Consortium, based on an earlier version by David Rosenthal of Sun Microsystems.

xpr — X Client

Print a dump of an X window
xpr [-append *filename***] [-compact] [-cutoff** *level***] [-density** *dpi***]**
[-device *devtype***] [-gamma** *correction***] [-gray] [-header** *string***]**
[-height *inches***] [-landscape] [-left** *inches***] [-noff]**
[-noposition] [-output *filename***] [-plane** *number***] [-portrait]**
[-psfig] [-render *algorithm***] [-rv] [-scale** *scale***]**
[-slide] [-split *n***] [-top** *inches***] [-trailer** *string***] [-width** *inches***] [***filename***]**

The X client **xpr** formats and prints a dump of a window that had been generated by the client **xwd**. It can format files for printing on PostScript printers, the Digital LN03 or LA100, the IBM PP3812 page printer, the Hewlett-Packard LaserJet (or other PCL printers), or the HP PaintJet.

By default, **xpr** prints the largest possible representation of the window on the output page. By setting the appropriate command-line options, you can add headers and trailers, specify margins, adjust the scale and orientation, and combine multiple window dumps to one output file.

Options

xpr recognizes the following command-line options:

-append *filename*
Append the output to *filename*. This option does not apply to PostScript printers.

-compact
Use simple, run-length encoding for compact representation of windows with lots of white pixels. This is supported only on PostScript printers. Note, too, that this option compresses white space but not black space, so it is not useful for reverse-video windows.

-cutoff *level*
Change the level of intensity by which **xpr** maps colors to black or white for monochrome output on a LaserJet printer. *level* gives the percentage of full brightness. Fractions are allowed.

-density *dpi*
Set to *dpi* the dot-per-inch density to be used by a Hewlett-Packard printer.

-device *devtype*
Print the output onto *devtype*, which must be one of the following:

la100 The Digital LA100.
ljet The Hewlett-Packard LaserJet series and other monochrome PCL devices, such as ThinkJet, QuietJet, RuggedWriter, and Hewlett-Packard 12560- and and 12930-series printers.
ln03 The Digital LN03.
lw The Apple LaserWriter. This is equivalent to option **pp**.
pjet The Hewlett-Packard PaintJet (color mode).
pjetxl The Hewlett-Packard PaintJet XL Color Graphics Printer (color mode).
pp The IBM PP3812.
ps Any PostScript printer.

The default is **ps**.

-gamma *correction*
Change the intensity of the colors printed by PaintJet XL printer. *correction* is a floating-point value in the range 0.00 to 3.00. Consult the operator's manual to determine the correct value for the specific printer.

-gray 2|3|4
Uses a simple 2×2, 3×3, or 4×4 gray-scale conversion on a color image, rather than mapping to strictly black and white. This doubles, triples, or quadruples the effective width and height of the image. This is not supported on IBM printers, or printers that run PCL.

-header *string*
Set a header string to be printed above the window.

-height *inches*
Set the maximum height of the page.

-landscape
Print the window in landscape mode. By default, **xpr** prints a window such that its longest side follows the long side of the paper.

-left *inches*
Set the left margin, in inches. Fractions are allowed. By default, the window is centered in the page.

-noff When specified with **-append**, this window appears on the same page as the previous window. This option does not apply to PostScript printers.

-noposition
Bypass generation of commands for header, trailer, and image positioning for LaserJet, PaintJet, and PaintJet XL printers.

-output *filename*
Write the output into *filename*. If this option is not specified, **xpr** writes its output to the standard output.

-plane *number*
Set the bit plane to use in an image. The default is to use the entire image and map values into black-and-white, based on color intensities.

-portrait
Print the window in portrait mode. By default, a window is printed such that its longest side follows the long side of the paper.

-psfig Suppress translation of the PostScript picture to the center of the page.

-render *algorithm*
Use *algorithm* to allow the Hewlett-Packard PaintJet XL printer to render the image with the best ratio of quality versus performance. Consult the operator's manual to determine which *algorithm*s are available.

-rv Print the window in reverse video.

-scale *scale*
Change the size of the window on the page. The PostScript, LN03, and Hewlett-Packard printers can translate each bit in a window pixel map into a grid of a specified size. For example, to translate each bit into a 3×3 grid, use the option:

```
-scale 3
```

By default, **xpr** prints a window with the largest scale that will fit onto the page for the specified orientation.

-slide Print overhead transparencies on the PaintJet and PaintJet XL printers. This is not supported on LaserJet printers.

-split *n* Split the window output over *n* pages. This might be necessary for very large windows; otherwise, cause the printer to overload and print the page in an obscure manner. This is not supported for PCL or PostScript printers.

-top *inches*
Set the top margin for the picture, in inches. Fractions are allowed.

-trailer *string*
Set a trailer string to be printed below the window.

-width *inches*
Set the maximum width of the page.

DEC Printers

The current version of **xpr** can print on the LN03 most X Windows that are not larger than two-thirds of the screen. The LN03 has memory limitations that can cause it to incorrectly print very large or complex windows. The two most common errors encountered are

```
band too complex
```

and:

```
page memory exceeded
```

In the case of the first error, a window may have a particular six-pixel row that contains too many changes (from black to white to black). This causes the printer to drop part of the line and possibly parts of the rest of the page. The printer will flash the number **1** on its front panel when this problem occurs. A possible solution to this problem is to increase the scale of the picture, or to split the picture onto two or more pages. The second problem occurs when the picture contains too much black, or if the picture contains complex half-tones such as the background color of a display. When this problem occurs, the printer will automatically split the picture into two or more pages. It may flash the number **5** on its from panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or to rework the application to produce a less complex picture.

There are several limitations on the LA100 support: the picture is always printed in portrait mode, there is no scaling, and the aspect ratio will be slightly off.

Hewlett-Packard Printers

If the command line does not include the option **-density**, **xpr** uses 300 does per inch (dpi) for the LaserJet and 90 dpi for the Pjet. Allowable values for the option **-density** are 300, 150, 100, and 75 dpi. Consult the operator's manual to determine densities supported by other printers.

If the command line does not specify the option **-scale**, **xpr** expands the image to fit the printable page area.

The default printable page area is 8×10.5 inches. Other paper sizes can be accommodated by using the command-

line options **-height** and **-width**.

Note that a 1024×768 image fits the default printable area when processed at 100 dpi with **-scale** set to one. You can print the same image using 300 dpi with **-scale** set to three, but this requires that considerably more data be sent to the printer.

You can tailor **xpr** to drive monochrome PCL printers other than the LaserJet. To print on a ThinkJet 12225A, invoke **xpr** with the following command:

```
xpr -density 96 -width 6.667 filename
```

Likewise, to print black-and-white output on a PaintJet, use the command:

```
xpr -density 180 filename
```

The monochrome intensity of a pixel is computed as 0.30×R + 0.59×G + 0.11×B. If a pixel's computed intensity is less than the level set by the option **-cutoff**, **xpr** prints it as white. This maps light-on-dark display images to black-on-white hard copy. The default cutoff intensity is 50% of full brightness. For example, specifying **-cutoff 87.5** moves the white/black intensity point to 87.5% of full brightness.

A LaserJet printer must be configured with sufficient memory to handle the image. A full page at 300 dots per inch requires approximately two megabytes of printer memory.

xpr produces color images on PaintJet at 90 dpi. The PaintJet is limited to 16 colors from its 330-color palette on each horizontal print line. **xpr** issues a warning message if it encounters more than 16 colors on a line. **xpr** programs the PaintJet for the first 16 colors it encounters on each line, and uses the nearest matching programmed value for other colors present on the line.

Specifying the reverse-video option **-rv** for the PaintJet interchanges black and white on the output image. No other colors are changed.

Multiplane images must have been recorded by **xwd** in **ZPixmap** format. Single-plane (monochrome) images may be in either **XYPixmap** or **ZPixmap** formats.

Some PCL printers do not recognize image-positioning commands. Output for these printers will not be centered on the page and header, and trailer strings may not appear where expected.

Example

Each image in this manual was captured from the screen by the X client **xwd**, then turned into an encapsulated PostScript file with the following command:

```
xpr -portrait -psfig -scale 3 file > file.eps
```

where *file* names the file into which **xwd** had written its output.

The scaling factor was necessary to size the window correctly. Note that a scaling factor of two was used with images of the root window (such as the one that appears in the Lexicon entry for **xsetroot**); otherwise, the image would not have fit into the format of this manual.

The PostScript output of **xpr** had to be edited slightly, to remove the instruction **showpage** at the bottom of the file. Note, too, that the bounding-box comment at the beginning of the file was extremely inaccurate; in some cases, the bounding box claimed that the image had a negative width or height. The only way to tell for certain is to print the image and measure it with a pica rule. In this manual, the final images are incorporated into the pages using custom-written **troff** macros.

See Also

X clients, xwd, xwud

Notes

xpr was written by Michael R. Gretzinger and Jose Capo of MIT Project Athena, Marvin Solomon of the University of Wisconsin, Bob Scheifler of MIT, Angela Bock and E. Mike Durbin of Rich Inc., and Larry Rupp of Hewlett-Packard.

xprop — X Utility

Display an application's properties

xprop [-help] [-grammar] [-id *id***] [-root] [-name** *name***] [-frame] [-font** *font***] [-display** *display***] [-len** *n***]**
[-notype] [-fs *file***] [-remove** *property-name***] [-spy] [-f** *atom format* **[***dformat***]]**

The X utility **xprop** displays the contents of properties. You can select a window or font by using an command-line argument or, in the case of a window, by clicking on it. **xprop** then displays that window's or font's properties, possibly with formatting information.

Options

xprop recognizes the following command-line options:

-display *host***[:***server***][.***screen***]**
Display the output window on screen number *screen* of *server* on host system *host*.

-f *name format* [*dformat*]
Use *format* as the format for property *name*, and use *dformat* as its dformat. If *dformat* is missing, **xprop** assumes **= $0+\n**.

-font *name*
Display the properties of font *name*.

-frame When the user selects a window interactively (i.e., if command-line options **-name**, **-root**, and **-id** are not used), look at the window manager's frame (if any) instead of looking for the client window.

-fs *file* Read *file* as a source of formats for properties.

-grammar
Print a detailed grammar of all command-line options.

-help Print a summary of command-line options.

-id *id* Display the properties of window with X identifier *id*. This is useful when the target window is not mapped to the screen or when using the mouse is impossible or would interfere with the application being examined.

-len *n* Read or display no more than *n* bytes of any property. This is useful when displaying the contents of the cut buffer, which can be several pages long.

-name *name*
Display the properties of the window named *name*.

-notype
Do not display a property's type.

-remove *property*
Remove *property* from the indicated window.

-spy Examine a window's properties forever, looking for changes in properties.

-root Display the properties of the X server's root window. This is useful in situations where the root window is completely obscured.

Selecting a Font or Window

To display the properties of a font, you must use the command-line option **-font**. To display the properties of a window, you can select the window in any of four ways:

1. If the desired window is the root window, use the command-line argument **-root**.
2. If the desired window is not the root window, you can selected it by its ID number, by using the command-line option **-id**.
2. If the desired window is not the root window, you can selected it by its name, by using the command-line option **-name**.
4. If you do not use any of the options **-font**, **-id**, **-name**, or **-root**, **xprop** displays a cross-hairs mouse cursor. To select a window, move the cursor to that window and click any button on the mouse.

By default, **xprop** prints first the property's name, then its type (if it has one) in parentheses, then its value. The command-line argument **-notype** tells **xprop** not to display a property's type. Argument **-fs** names a file that contains a list of formats for properties, and the argument **-f** specifies the format for one property.

Formats

The formatting information for a property consists of two parts: *format* and *dformat*. The former specifies the format of the property itself (i.e., whether it made up of words, bytes, or **long**s, etc.), whereas the latter specifies how the property should be displayed. If you do not give **xprop** a list of properties, it prints information about every property of the selected window or font. If a window or font does not contain a requested property, **xprop** prints

```
not defined
```

for that property.

The following paragraphs describe how to construct *format* and *dformat* strings. However, for the vast majority of uses, this should not be necessary: the built-in defaults are sufficient to display all standard properties. You should find it necessary to specify a *format* and *dformat* only if you dislike the standard display format. We encourage new users, in particular, to skip this part.

A *format* begins with a number that gives the number of bits per field in this property. **xprop** recognizes the following numbers:

0 Use the field-size information within the property itself. This is only needed for special cases like type **INTEGER**, which is actually three different types, depending upon the size of the fields of the property.

8 The property consists of a sequence of bytes.

16 The property is a sequence of words. The difference between this and a sequence of bytes is the machine of the opposite byte ordering will swap the bytes of a word, where it will leave a sequence of bytes alone. (For more information on byte ordering, see the entries for **byte ordering** and **swab()** in the COHERENT Lexicon.)

32 The property consists of a sequence of long words.

Once the size of the fields has been specified, you must specify the type of each field. This is done using one format character per field. If you supply fewer format characters than the property has fields, **xprop** repeats the last format character as many times as necessary for the extra fields. The format characters and their meaning are as follows:

a The field holds an atom number. A field of this type should be of size 32.

b The field is Boolean. A '0' means "false," whereas anything else means "true".

c The field is a cardinal value, i.e., an unsigned integer.

i The field is a signed integer.

m The field is a set of bit flags; '1' means "on".

s The field is a string. **xprop** follows the C standard, in that it assumes that all subsequent bytes are characters until it reads a NUL (that is, a character with value zero). Use this format character only with a field size of **8**.

x The field is a hexadecimal number. It is the same as the formatting character **c**, except that the output is displayed in hexadecimal notation. This is most useful for displaying window identifiers and the like.

For example, consider the format **32ica**. This describes a property that consists of three 32-bit fields, the first of which holds a signed integer, the second an unsigned integer, and the third an atom.

A **dformat** defines how you want **xprop** to display information about the property in question. It cannot begin with a letter or a hyphen, so **xprop** can distinguish it from a property name or an argument. For example, the *dformat*

```
" is ( $0, $1 \)\n"
```

renders **POINT 3, -4** (which has a *format* of **32ii**)**as:**

```
is ( 3, -4 )\n
```

xprop prints literally any character other than **$**, **?**, ****, or **(**. To print a literal **$**, **?**, ****, or **(**, precede it by with a ****. **xprop** also recognizes the following escape sequences:

\n Print a newline character.

\t Print a tab character.

*****O* Print the number *O* in octal notation.

The notation **\$***N* tells **xprop** to display the contents of field number *N*. How **xprop** formats the displayed field depends on the field's format character in the corresponding *format* string. For example, if the *format* string defines a character s being of type **c** (that is, a cardinal), **xprop** prints it in decimal; whereas if the *format* defines it as being of type **x** (that is, a hexadecimal number), **xprop** displays it in hexadecimal notation.

If the field is not present in the property (which possible with some properties), **xprop** prints:

```
<field not available>
```

The escape sequence **$***n***+** displays every field from *n* through the end of the property. If field *n* is not defined, **xprop** displays nothing. This is useful for a property that is a list of values.

The formatting sequence **?***exp(text***)** displays *text* only if expression *exp* evaluates to non-zero. This is useful for two reasons. First, it allow you display a field only if a flag is set. Second, it lets you display a value (such as a state number) as a name rather than as a number.

The syntax of *exp* is as follows:

exp **::=** *term* **|** *term***=***exp* **|** **!***exp*
term **::=** *n* **|** **$***n* **|** **m***n*

The symbol **!** is a logical NOT: it changes zero to one, and any non-zero value to zero. The symbol **=** is the equality operator. Note that **xprop** evalutes all expressions as 32-bit numbers, so -1 does not equal 65,535. **=** returns one if its two values are equal, and zero if they are not. *n* represents the constant value *n*, whereas **$***n* represents the value of field *n*. **m***n* is one if flag *n* in the first field with format character **m** in the corresponding *format* string has value one; otherwise, it is zero.

For example, the format

```
?m3(count: $3\n)
```

displays field 3 with a label of **count** only if flag 3 is on. (Note that flags are counting beginning with 0.) The format

```
?$2=0(True)?!$2=0(False)
```

displays the string **True** if field 2 equals zero, and displays **False** if field 2 equals anything other than zero.

To display a property, **xprop** needs both a *format* and a *dformat*. Before **xprop** uses its default values of a *format* of **32x** and a *dformat* of

```
= { $0+ }\n
```

it searches several places to find more specific formats. First, it searches using the name of the property. If this fails, it searches using the type of the property. This lets you give properties of type **STRING** with one format, yet give property **WM_NAME** (which is of type **STRING**) a different format.

The locations searched are in the following order:

- The format, if any is specified with the property name.
- The formats defined by the options **-f**, in last to first order.
- The contents of the file named by the command-line option **-fs**.
- The contents of the file specified by the environmental variable **XPROPFORMATS**, if any.
- Finally, **xprop**'s built-in formats.

The files referred to by the **-fs** argument and the variable **XPROPFORMATS** consist of one or more lines of the following form:

name format [*dformat*]

where *name* names a property or type. If *dformat* is not present, **xprop** uses

```
" = $0+\n"
```

by default.

Examples

To display the name of the root window, use the command:

```
xprop -root WM_NAME
```

To display the window manager's hints for the clock, use the command:

```
xprop -name xclock WM_HINTS
```

To display the first 100 bytes of the cut buffer, use the command:

```
xprop -root -len 100 CUT_BUFFER0
```

To display the point size of the fixed font, use the command:

```
xprop -font fixed POINT_SIZE
```

Finally, to display all the properties of window 0x200007, use the command:

```
xprop -id 0x200007
```

Environment

xprop reads the following environmental variables:

DISPLAY
The default display.

XPROPFORMATS
The name of the file from which additional formats are obtained.

See Also

appres, X utilities, xwininfo

Notes

xprop was written by Mark Lillibridge of MIT Project Athena.

xrdb — X Utility

Read/set the X server's resource data base
xrdb [-*option ...***] [***file***]**

xrdb manipulates the X server's resource data base. Clients, whatever their system of origin, read this data base to set their resources; its contents overrides any resources set in an application's defaults file. This lets you change defaults dynamically, without editing defaults files.

The X server stores its resource data base in the property **RESOURCE_MANAGER** on the root window of screen 0 (the default screen used by the server), or the property **SCREEN_RESOURCES** on the root window of any or all screens, or everything combined. Most X clients read the properties **RESOURCE_MANAGER** and **SCREEN_RESOURCES** to get your preferences concerning color, fonts, and other properties. **RESOURCE_MANAGER** holds resources that apply to all screens of the display. **SCREEN_RESOURCES** on each screen specifies additional (or overriding) resources to be used for that screen. (Because X Windows for COHERENT presently supports only one screen, so **SCREEN_RESOURCES** is not used — all resources are placed into **RESOURCE_MANAGER**.)

xrdb passes the contents of the file specified by *file* (or the data read from standard input if the file name '-' is used), through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

BITS_PER_RGB=*num*
The number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it usually is not related to **PLANES**.

CLASS=*visualclass*
Set the class of display, where *visualclass* is one of the following: **StaticGray**, **GrayScale**, **StaticColor**, **PseudoColor**, **TrueColor**, or **DirectColor**. This is the visual class of the root window of the default screen.

COLOR
This is defined only if **CLASS** is **StaticColor**, **PseudoColor**, **TrueColor**, or **DirectColor**.

HEIGHT=*num*
The height of the default screen, in pixels.

SERVERHOST=*hostname*
The host-name portion of the display to which you are connected.

HOST=*hostname*
Same as **SERVERHOST**.

CLIENTHOST=*hostname*
The name of the host on which **xrdb** is running.

PLANES=*num*
The number of bit planes (the depth) of the root window of the default screen.

RELEASE=*num*
The vendor's release number for the server. The interpretation of this number will vary depending on **VENDOR**.

REVISION=*num*
The X protocol's minor version supported by this server.

VERSION=*num*
The X protocol's major version supported by this server (should always be 11).

VENDOR=*vendor*
A string that names the vendor of the server.

WIDTH=*num*
The width of the default screen, in pixels.

X_RESOLUTION=*num*
The X resolution of the default screen, in pixels per meter.

Y_RESOLUTION=*num*
The X resolution of the default screen, in pixels per meter.

xrdb ignores all lines that begin with an exclamation mark '!'. You can use them as comments.

Because **xrdb** can read the standard input, you can use it to redefine a property directly from a terminal or from a shell script.

Options

xrdb recognizes the following command-line options:

-all Manipulate the screen-independent resource property (**RESOURCE_MANAGER**) as well as the screen-specific property (**SCREEN_RESOURCES**) on every screen of the display. For example, when used with option **-query**, **xrdb** prints the contents of all properties. For the options **-load** and **-merge**, **xrdb** processes the input file once for each screen. In its output, **xrdb** gathers all resources that occur in common, and applies them as the screen-independent resources. It applies the remaining resources for each per-screen property. This the default mode of operation.

-backup *string*
Append *string* to the end of each file name used with option **-edit**, to generate a backup file.

-cpp *name*
Name the preprocessor to use. *name* should be its full path name. Although **xrdb** was designed to use **cpp**, you can use any program that acts as a filter and accepts the options **-D**, **-I**, and **-U**.

-D*symbol***[=***value***]**
Tell **cpp** to define *symbol* and give it *value*.

-display *display*
Set the X server to be used. This option also specifies the screen to use for the option **-screen**, and specifies the screen from which preprocessor symbols are derived for the option **-global**.

-edit *file*
Write the specified properties into *file*, replacing any values therein. This lets you to fold any changes that you have made to your defaults into your resource file, while preserving any comments or preprocessor lines.

-help Print a brief summary of these options.

-global Perform the operation only on the screen-independent property **RESOURCE_MANAGER**.

-I*directory*
Tell **cpp** to search *directory* for header files.

-load Load the input as the new values of the specified properties, replacing its current contents. This is the default action.

-merge Merge the input into the specified properties, rather than replacing their contents. This option performs lexicographically sorted merge of the two inputs, which is almost certainly not what you want but remains for backward compatibility.

-n When used with the options **-load** or **-merge**, show on the standard output all changes to the specified properties, but do not execute them. When used with the option **-edit**, show but do not execute all changes to the resource file.

-nocpp Do not filter the input file through a preprocessor before loading it into properties.

-query Print onto the standard output the contents of the specified properties. Because preprocessor commands in the input resource file are part of the input file, not part of the property, they will appear in the output that this option triggers. Use the option **-edit** to merge the contents of properties back into the input-resource file without damaging preprocessor commands.

-quiet Do not display warning messages about duplicate entries.

-remove
Remove the specified properties from the server.

-retain Tell the server not to reset itself if **xrdb** is the first client. This is never necessary under normal conditions, because **xinit** always acts as the first client.

-screen
Perform the operation only on the property **SCREEN_RESOURCES** on the default screen of the display.

-screens
Perform the property **SCREEN_RESOURCES** for each screen of the display. With the options **-load** and **-merge**, **xrdb** processes the input file for each screen.

-symbols
Print onto the standard output all symbols that are defined for the preprocessor.

-U*symbol*
Tell **cpp** to un-define *symbol*.

Environment

xrdb reads the environmental variable **DISPLAY** to find which display to use.

Using xrdb

As noted above, **xrdb** reads the *file* named on its command line. The X display manager **xdm**, which is not included with X Windows for COHERENT, invokes **xrdb** automatically to read the file **$HOME/.Xdefaults**. If you wish to use **xrdb** under **xinit**, which is the standard way of invoking X under X Windows for COHERENT, you must invoked from within the file **$HOME/.xinitrc**. For example

```
xrdb -load < $HOME/.Xdefaults
```

loads your personal defaults file into the X server.

The tutorial *X Windows Clients*, which appears earlier in this manual, gives examples of how to write a resource file

LEXICON

for **xrdb**.

See Also

appres, editres, X utilities

Notes

xrdb was written by Bob Scheifler and Phil Karlton, based on the original program written by Jim Gettys.

xrefresh — X Utility

Refresh all or part of an X screen
xrefresh [-*option ...***]**

The X utility **xrefresh** repaints all or part of your screen. This is useful when the screen has in some way become confused, e.g., by system messages. It maps a window on top of the desired area of the screen, then immediately unmaps it, which forces the X server to send refresh events to all applications.

By default, **xrefresh** uses a window with no background, which causes all applications to repaint "smoothly." However, you can use various options instead a solid background (of any color) or the root window's background.

Options

xrefresh recognizes the following command-line options:

-black Use a black background — in effect, turn off all of the electron guns within the tube. This can be somewhat disorienting as the screen goes black for a moment.

-display *host***[:***server***][.***screen***]**
Refresh *screen* of *server* on host system *host*.

-none This is the default. **xrefresh** repaints all windows.

-root Use the root window's background.

-solid *color*
Use a solid background of *color*. Try **green**.

-white Use a white background. The screen paints itself in white, then immediately repaints the root window's background and all application windows.

X Defaults

xrefresh uses the routine **XGetDefault()** to read defaults, so the names of all of its resourcs are capitalized:

Black
None
Root
Solid
White Determine what to paint the window, before repainting the background and applications' windows.

Geometry
Determine the area to refresh. This option is not very useful.

Environment

xrefresh reads the environmental variable **DISPLAY** to get host and display number.

See Also

X utilities

Notes

xrefresh was written by Jim Gettys of Digital Equipment Corporation and MIT Project Athena.

xset — X Utility

Set preferences for the display

xset [-b] [-b on/off] [-b [*volume* **[***pitch* **[***duration***]]] [[-]bc] [-c] [-c on/off] [-c [***volume***]]**
[-display *display***] [[-+]fp[-+]** *directory***[,***directory***[, ...]]] [-fp default] [-fp rehash]**
[[-]led [*integer***]] [led on/off] [-mouse [***accel_mult***[/***accel_div***] [***threshold***]]]**
[-mouse default] [-P [*period***]]] [-p** *pixel color***] [-q] [[-]r] [-r on/off] [-s [***length***]]**
[-s blank/noblank] [-s expose/noexpose] [-s on/off] [-s default]

The X utility **xset** lets you set options for the display. It recognizes the following command-line options:

[-]b [*volume [pitch* **[***duration***]]]**
Set the volume on the bell, both pitch and duration. This option accepts up to three numerical parameters, a preceding hyphen '-', or an **on/off** flag. If the command line gives no parameters or the flag **on**, **xset** uses the system defaults. If the command includes a hypen or or the flag **off**, **xset** turns the bell off. *volume* gives the volume of the bell, as a percent of its maximum volume. *pitch* gives the bell's pitch, in hertz. *duration* gives the bell's the duration, in milliseconds. Not all hardware can vary the bell's characteristics. The X server sets the characteristics of the bell as closely as it can, but it may not be able to set them exactly to your specifications.

[-]bc Set the bug-compatibility mode in the server, if possible. A hyphen '-' disables the mode; otherwise, the mode is enabled. This option explicitly reintroduces certain bugs into the X server, so that X can run many clients that compensated for bugs in releases that preceded release 4. Use this mode with care: new applications should be developed with it disabled. For this option to work, the server must support the protocol extension **MIT-SUNDRY-NONSTANDARD**.

[-]c [*value***]**
Control key-click. A hyphen or an **off** flag disables this option, which turns off the key-click. *value* sets the volume of the keyclick, from zero to 100; the X server sets the volume to the nearest level that the hardware can support.

-display *server*
Set the configuration of display *server*.

fp=*directory***[,...]**
Set the font path: the X server searches each *directory* for fonts. The server ignores each *directory* that does not contain a data base created by **mkfontdir**.

fp default
Set the font path to the server's default.

fp rehash
Re-read the font data bases in the current font path. Use this only after you have added new fonts to a font directory and have run **mkfontdir** to add the fonts to that directory's font data base.

-fp *directory***[,** *...]*
fp- *directory***[,** *...]*
Remove each *directory* from the font path.

+fp *directory***[,***directory***, ...]]**
fp+ *directory***[,***directory***, ...]]**
Append each *directory* to, respectively, the beginning or end of the font path.

[-]led [*value***]**
Toggle the keyboard LEDs, that is, the little lights that indicate whether the **NumLock**, **CapsLock**, and **ScrollLock** keys are set. If no *value* is set or the flag **on** is set, turn on all LEDs. If a preceding hyphen or the flag **off** is set, turn off all LEDs. Setting *value* to between one and three turns the corresponding LED on or off, depending on whether this is preceded by a hyphen. Different *value*s may refer to different LEDs on different hardware.

m [*acceleration* **[threshold]]**
m default
This option sets mouse values. *acceleration* sets the mouse-acceleration: it can be an integer or a simple fraction. The mouse goes *acceleration* times as fast when it travels more than *threshold* pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet can be set to travel across the screen in a flick of the wrist when desired. Using the flag **default** or omitting all

arguments sets the mouse parameters to the system's default.

p [*entry* **[***color***]]**
Set the pixel color values. *entry* gives color map's entry number, in decimal; *color* gives a color specification. On some servers, you can change the background color on the root window by altering the entries for resources **BlackPixel** and **WhitePixel**. Although these are often zero and one, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.

[-]r Control the autorepeat. If a hyphen precedes this option, or if it is followed by the flag **off**, **xset** turns autorepeat off. **xset** enables autorepeat if this option is followed by the flag **on**, or is followed by no flags at all.

s [*time***]***fB***] [***cycle***] [blank | noblank] [expose | noexpose] [on | off] [default]**
Set screen-saver parameters, as follows:

time Wait *time* seconds before activating the screen saver.
cycle Modify the screen-saver's pattern after *cycle* seconds, to prevent the screen from burning in.

blank If the screen-saver option is set, blank the screen.

noblank
If the screen-saver option is set, display a pattern of some sort.

expose Allow window exposures.

noexpose
Disable the screen saver unless the server can regenerate the screens without causing exposure events.

on Turn on the screen saver.

off Turn off the screen saver.

default Return parameters to their default values.

q Print information on the current settings.

The X server resets these values to their defaults after you log out.

Example

The following gives example output of the command **xset q**:

```
Keyboard Control:
  auto repeat:  on    key click percent:  0    LED mask:  00000000
  auto repeating keys:  0000000000000000
                        0000000000000000
                        0000000000000000
                        0000000000000000
  bell percent:  50    bell pitch:  400    bell duration:  100
Pointer Control:
  acceleration:  2/1    threshold:  4
Screen Saver:
  prefer blanking:  yes    allow exposures:  yes
  timeout:  600    cycle:  600
Colors:
  default colormap:  0x21    BlackPixel:  0    WhitePixel:  1
Font Path:
  /usr/X11/lib/fonts/misc/,/usr/X11/lib/fonts/75dpi/
Bug Mode: compatibility mode is disabled
```

See Also

X, xmodmap, X utilities, xrdb, xsetroot

Notes

xset was written by Bob Scheifler of the MIT Laboratory for Computer Science, and David Krikorian of MIT Project Athena.

xsetroot — X Utility

Set preferences for the root window

xsetroot [-bg *color***] [-bitmap** *filename***] [-cursor** *cursorfile maskfile***] [-cursor_name** *cursorname***]**
[-def] [-display *display***] [-fg** *color***] [-gray] [-grey] [-help] [-mod** *x y***]**
[-name *string***] [-rv] [-solid** *color***]**

xsetroot modifies the appearance of the screen's background — what is usually called the *root window*. **xsetroot** lets you tailor the appearance of the root window to suite your tastes.

Options

If you invoke **xsetroot** with no command-line options, it resets the root window to its default settings. **xsetroot** recognizes the following command-line options:

-bg *color*
Set the background to *color*.

-bitmap *file*
Use the bit map specified in *file* to tile the background. The entire background is made up of repeated "tiles" of the bit map.

-cursor *cursorfile maskfile*
Change the mouse cursor that X displays when the cursor is against the root window. *cursorfile* and *maskfile* name bit-mapped images. For example, the command

```
xsetroot -cursor /usr/X11/include/X11/bitmaps/star \
      /usr/X11/include/X11/bitmaps/starMask
```

changes the mouse cursor to an asterisk when the cursor is against the root window.

-cursor_name *cursorname*
Change the mouse cursor to one of the standard cursors from the **cursor** font. File **/usr/X11/include/X11/cursorfont.h** names the available cursors (you should ignore the string **XC_** that prefixes each name). The Lexicon article for the X utility **xfd** displays the cursors in that font.

-def Reset all unspecified attributes to their default values. This restores to the familiar gray mesh and the cursor to the hollow × shape. If you specify **-def** with other options, **xsetroot** sets only the non-specified characteristics to their default states.

-display *server*
Connect to *server*.

-fg *color*
Set foreground to *color*. Foreground and background colors are meaningful only in combination with the options **-cursor**, **-bitmap**, and **-mod**.

-gray
-grey Make the entire background gray.

-help Print a usage message and exit.

-mod *x y*
Draw a plaid-like grid pattern on the screen. *x* are integers, ranging from one to 16, that give the number of pixels that separate, respectively, the vertical and horizontal lines.

-rv Exchange the foreground and background colors, to mimic reverse video. Normally, the foreground color is black and the background color is white.

-solid *color*
Set the background of the root window to *color*. The file **/usr/X11/lib/rgb.txt** names all of the colors that X recognizes.

You can use only one of the options that set the background of the root window, i.e., **-solid**, **-gray**, **-grey**, **-bitmap**, and **-mod**.

Example

The following command tiles the background of the root window with a bit map:

```
xsetroot -bitmap /usr/X11/include/X11/bitmaps/escherknot
```

When you type it, your screen appears something like the following:

See Also

xset, xrdb, X utilities

Notes

xsetroot was written by Mark Lillibridge of MIT Project Athena.

xstdcmap — X Utility

X standard color-map utility

xstdcmap [-all] [-best] [-blue] [-default] [-delete *map***] [-display** *display***] [-gray] [-green] [-help] [-red] [-verbose]**

The X utility **xstdcmap** defines the properties of the standard color map. You should invoke it from within the script **$HOME/.xinitrc**, to create the standard color-map definitions, which facilitates sharing of scarce color-map resources among clients. Where at all possible, the color maps it creates have read-only allocations.

Options

xstdcmap recognizes the following command-line options:

-all Define all six standard color-map properties on each screen of the display. Not every screen supports visuals under which all six standard color-map properties are meaningful. **xstdcmap** determines the best allocations and visuals for the color-map properties of a screen. Any previously existing standard color-map properties will be replaced.

-best Define **RGB_BEST_MAP**.

-blue Define **RGB_BLUE_MAP**.

-default Define **RGB_DEFAULT_MAP**.

-delete *map*
Remove a standard color-map property. *map* can be one of the following: **default**, **best**, **red**, **green**, **blue**, or **gray**.

-gray Define **RGB_GRAY_MAP**.

-green Define **RGB_GREEN_MAP**.

-help Print on the standard error a brief description of these options.

-red Define **RGB_RED_MAP**.

-verbose Tell **xstdcmap** to print logging information as it parses its input and defines the standard color-map properties.

Environment

xstdcmap reads the environmental variable **DISPLAY** to find the default host and display number.

See Also

X utilities

Notes

xstdcmap was written by Donna Converse of the MIT X Consortium.

xterm — X Client

Terminal emulator for X
xterm [-*option ...* **]**

xterm is a terminal emulator for the X Window System. It opens a window that mimics either the DEC VT102 or the Tektronix 4014 terminals. You can type text into this window, just as if you were at a terminal, and so invoke X clients or utilities, or work with COHERENT commands that do not use the X Window System directly.

When you resize the **xterm**'s window, it sends the signal **SIGWINCH** to notify any child COHERENT programs that the size of the window (in effect, the size of the terminal) has changed. Programs that understand that signal can then resize themselves to work with the new window dimensions. Note that as of this writing, most COHERENT applications do not yet understand how to work with **SIGWINCH**.

The VT102 and Tektronix 4014 terminals each has their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (that is, height versus width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit into the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor. The active window can be chosen through escape sequences, through the menu **VT Options** in the VT102 window, or through the menu **Tek Options** in the 4014 window.

Emulations

xterm's emulation of the VT102 is fairly complete, but does not support the blinking-character attribute nor the double-wide and double-size character sets. **termcap** entries that work with **xterm** include **xterm**, **vt102**, **vt100**, and **ansipc**. **xterm** automatically searches the file **/etc/termcap** for these entries, and then sets the environmental variable **TERM**. You can modify many of **xterm**'s special features (like logging) by typing a set of escape sequences different from the standard VT102 escape sequences.

xterm's emulation of the Tektronix 4014 is also fairly good. It supports four sizes of font and five types of line. **xterm** internally records all Tektronix text and graphics commands; you can tell it write them into a file by sending it the escape sequence **COPY**, or through the **Tektronix** menu described below. The name of the file will be

COPY*yy-MM-dd.hh:mm:ss*

where *yy*, *MM*, *dd*, *hh*, *mm*, and *ss* give, respectively, the year, month, day, hour, minute, and second when **COPY** was executed. **xterm** the file into the directory from which **xterm** was launched, or the home directory for a login **xterm**.

Other Features

xterm automatically highlights the text cursor when the mouse cursor enters its window, and un-highlights it when the mouse cursor exits the window. If the **xterm** window is the screen's focus window, then the text cursor is highlighted no matter where the mouse cursor is.

When it is running in VT102 mode, **xterm** has escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When this buffer is activated, **xterm** saves the current screen and replaces it with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The **termcap** entry for **xterm** allows the editor **vi** to switch to the alternate screen for editing and to restore the screen on exit.

Both VT102 or Tektronix modes support escape sequences to change the name of the windows.

Options

xterm recognizes the following command-line options. Prefixing an option with '+' instead of '-' returns the option to its default value:

-132 Recognize the VT102 escape sequence **DECCOLM**, which switches between 80 and 132 columns, and resize the window appropriately. By default, **xterm** ignores this escape sequence.

-ah Always highlight the text cursor. By default, **xterm** displays a hollow text cursor whenever the mouse cursor is not in the **xterm** window or its icon.

-aw Allow auto-wraparound. This feature lets the text cursor wrap automatically to the beginning of the next line when when it is at the end of a line and more text is output.

+aw Do not allow auto-wraparound.

-b *pixels*
Set the width of the inner border (that is, the distance between the outer edge of the text characters and the window's border) to *pixels*. The default is two.

-bd *color*
Set the color of the border to *color*. The default is **black**.

-bg *color*
Set the color of the background to *color*. The default is **white**.

-bw *pixels*
Set the width of the border to *pixels*.

-C This window should receive text directed to **/dev/console**. This is not supported on all systems. To obtain console output, you must own the console device and have read and write permissions for it.

-cb Set the **vt100** resource **cutToBeginningOfLine** to **FALSE**.

+cb Set the **vt100** resource **cutToBeginningOfLine** to **TRUE**.

-cc *characterclassrange***:***value***[,...]**
Use the classes indicated by the given ranges when selecting by words. See the section on character classes, below.

-cn Do not cut newlines in line-mode selections.

+cn Cut newlines in line-mode selections.

-cr *color*
Set the color of the text cursor to *color*. The default is to use the same foreground color as that used for text.

-cu Work around a bug in **curses** that causes COHERENT application **more** to drop the leading tab from a line that begins with a tab and is preceded by a line that is exactly the width of the window.

-cu Do not work around the above-described bug in **curses**.

-e *program* **[***argument* **[, ...]]**
Run *program* with each *argument* in an **xterm** window. If the command line gives neither the option **-T** nor the option **-n**, **xterm** sets the window title and icon name the base name of *program*. *This must be the last option on the command line.* This option often is used to invoke **xterm** while running a shell other than your default shell.

-fb *font* Display boldfaced text in *font*, which must the same height and width as the normal font. If the command line specifies only the normal font or the boldface font, **xterm** uses that font as the normal font and produces boldface by overstriking it. The default is to overstrike the normal font.

-fg *color*
Set the color of the foreground to *color*. The default is **black**.

-fn *font* Use *font* in the display. The default is **fixed**.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-help Print a message that describes these options.

-iconic Begin the window as an icon rather than as a normal window.

-im Turn on the resource **useInsertMode**.

+im Turn off the resource **useInsertMode**.

-j Perform jump scrolling. Normally, **xterm** scrolls text one line at a time; jump scrolling lets **xterm** scroll multiple lines at once, so it does not fall behind the program that is writing text to the window. This option makes **xterm** much faster when it scrolls through large amounts of text. You can also use the VT100 escape sequences and the menu **VT Options** to turn feature on or off.

+j Do not use jump scrolling.

-l Write all terminal output into a log file as well as onto the screen. You can also use the menu **VT Options** to enable or disable this option.

+l Do not log what is written to the screen.

-lf *file* Write text logged by the option **-l** into *file*. If *file* begins with a pipe symbol '|', **xterm** assumes that the rest of its command line is a shell command. The default log file is named **XtermLog.***XXXXX*, where *XXXXX* is the process id of **xterm**. By default, the log file is written into the directory from which **xterm** was launched; or in the case of a login window, into the user's home directory.

-ls Let the shell that is started in the **xterm** window be a login shell (i.e., the first character of **argv[0]** will be a hyphen, which tells the shell to execute your **.profile**).

+ls Do not let the shell that is started in the **xterm** window be a login shell.

-mb Ring the bell when the user typing nears the right margin of a line, as on an old-fashioned typewriter. You can also turn this option on or off from the menu **VT Options**.

+mb Do not ring the bell when the user's typing nears the margin. This is the default.

-mc *milliseconds*
Set to *milliseconds* the maximum time between multi-click selections.

-ms *color*
Set the mouse cursor to *color*. The default is to use the foreground color.

-name *name*
Obtain resources under *name* rather than the default, which is executable program's file name. *name* should not contain the characters '.' or '*'.

-nb *number*
Ring the bell when the text cursor is *number* characters from the right margin. The default is ten. This option is meaningful only if the option **-mb** is also set.

-rv Simulate reverse video by exchanging the foreground and background colors.

-rw Allow reverse-wraparound. This feature lets the test cursor back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines, and its use is encouraged. You can turn this option on or off from the menu **VT Options**.

+rw Do not allow reverse-wraparound.

-S*ccn* Set the last two letters of the name of a pseudoterminal to use in slave mode, plus the number of the inherited file descriptor. The option is parsed **%c%c%d**. This lets you use **xterm** as an input and output channel for an existing program.

-s Let **xterm** scroll asynchronously. This means that **xterm** does not have to keep the screen completely up to date as it scrolls. This lets **xterm** run faster when network latencies are very high, and typically is used when running across a very large network or across many gateways.

+s Force **xterm** to scroll synchronously.

-sb Permit **xterm** to save in a buffer a number of lines of text that have scrolled off the top of the window, and display a scrollbar that lets you access those lines of text interactively. The command-line option **-sl** sets the number of lines to save. You can turn this option on or off from the menu **VT Options**.

+sb Display a scrollbar for redisplaying old text.

-sf Generate Sun Function Key escape codes for function keys.

+sf Generate the standard escape codes for function keys.

-si Output to a window does not automatically reposition the screen to the bottom of the scrolling region. You can turn this option on or off from the menu **VT Options**.

+si Output to a window repositions the screen to the bottom of the scrolling region.

-sk Pressing a key while using the scrollbar to review old text automatically repositions the screen to the bottom of the scroll region.

+sk Pressing a key while using the scrollbar to review old text does not reposition the screen to the bottom of the scroll region.

-sl *number*
Buffer *number* lines that have scrolled off the top of the window. The default is 64.

-t Launch **xterm** in Tektronix mode rather than in VT102 mode. You can use the menu **Options** to switch between the modes.

+t Launch **xterm** in VT102 mode. This is the default.

-title *string*
Display *string* in the title area of the window. The default title is the command line specified after the command-line option **-e**, if any; otherwise, the application name is used.

-tm *command*=*keystrokes*[,...]
Invoke internal terminal *command* with *keystrokes*, similar to settings of the COHERENT command **stty**. **xterm** recognizes the following commands:

intr	**quit**	**erase**	**kill**
eof	**eol**	**swtch**	**start**
stop	**brk**	**susp**	**dsusp**
rprnt	**flush**	**weras**	**lnext**

You can specify a control charactger as **^***char* (e.g., **^c** or **^u**).

-tn *name*
Name the terminal type to be set in the environmental variable **TERM**. This terminal type must exist in the data bases **termcap** or **terminfo**, should include the entries **li#** and **co#**.

-ut Tell **xterm** not to write a record into the system log file **/etc/utmp**.

+ut Tell **xterm** to write a record into the system log file **/etc/utmp**.

-vb Use a visual bell rather than an audible one. When it reads the character **<ctrl-G>**, **xterm** briefly redraws the screen in reverse video instead of ringing the bell. Note that on the typical personal computer, this is rather slow.

+vb Do not use a visual bell. This is the default.

-wf **xterm** should wait for its window to be mapped for the first time before it start any subprocess (e.g., a shell or editor). This helps ensure that that application's initial settings for environment and terminal size are correct. It is the application's responsibility to catch subsequent changes to the terminal's size.

+wf **xterm** should not wait for its window to appear before it launches the subprocess.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Resources

In addition to all of the core X Toolkit resource names and classes, **xterm** also understands the following resource names and classes:

iconGeometry (class **IconGeometry)**
Set the preferred size and position of the application when it is iconified. It is not necessarily obeyed by all window managers.

iconName (class **IconName)**
Set the name to appear on the icon. The default is **xterm**.

termName (class **TermName)**
Set the terminal type to be set into the environment variable **TERM**.

title (class **Title)**
Set the string that the window manager can used when it displays this application.

ttyModes (class **TtyModes)**
Set the string that contains the terminal-setting commands and key sequences to which each is be bound. Same as the command-line option **-tm**, described above. This is very useful for overriding the default terminal settings without having to use the command **stty** every time you invoke **xterm**.

useInsertMode (class **UseInsertMode)**
Force the use of insert mode by adding appropriate entries to the environmental variable **TERMCAP**. This is useful if your system's **termcap** entry for this type of terminal is broken. The default is **false**.

utmpInhibit (class **UtmpInhibit)**
Specify whether **xterm** should try to record the user's terminal in **/etc/utmp**.

sunFunctionKeys (class **SunFunctionKeys)**
Specify whether to generate Sun Function Key escape codes for function keys, instead of standard escape sequences.

waitForMap (class **WaitForMap)**
Specify whether to wait for the initial window map before starting the subprocess. The default is **false**.

vt100 Resources

The following resources are specified as part of the **vt100** widget (class **VT100**):

allowSendEvents (class **AllowSendEvents)**
Specify whether to interpret or discard synthetic key and button events (generated using the X protocol **SendEvent** request). The default is **false**, which means they are discarded. Note that allowing such events creates a very large security hole.

alwaysHighlight (class **AlwaysHighlight)**
Specify whether **xterm** should always display a highlighted text cursor. By default, it displays a hollow text cursor whenever the mouse cursor moves out of the window or the window loses the input focus.

appcursorDefault (class **AppcursorDefault)**
If **true**, the mouse buttons are initially in application mode. The default is **false**.

appkeypadDefault (class **AppkeypadDefault)**
If **true**, the keypad keys are initially in application mode. The default is **false**.

autoWrap (class **AutoWrap)**
Specify whether to enable auto-wraparound. The default is **true**.

bellSuppressTime (class **BellSuppressTime)**
Set the number of milliseconds after the bell command during which **xterm** suppresses another bell command. The default is 200. This feature is most useful with the visible bell.

boldFont (class **BoldFont)**
Name the font to use for boldface, instead of the default overstriking.

c132 (class **C132)**
Specify whether to honor escape sequence VT102 DECCOLM. The default is **false**.

cutNewline (class **CutNewline)**
If **false**, triple-clicking to select a line does not include the newline character at the end of the line. If true, **xterm** also selects the newline. The default is **true**.

cutToBeginningOfLine (class **CutToBeginningOfLine)**
If **false**, triple-clicking to select a line selects only from the current word forward. If **true**, the entire line is selected. The default is **true**.

charClass (class **CharClass)**
Specify a comma-separated list of character-class bindings of the form **[***low***-]***high***:***value*. These determine which sets of characters should be treated identically when performing cut and paste. For details, see the section on character classes, below.

LEXICON

curses (class **Curses)**
Specify whether to work around the last-column bug in **curses**. The default is **false**.

background (class **Background)**
Specify the color to use for the background of the window. The default is **white**.

foreground (class **Foreground)**
Specify the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the text color change color. The default is **black**.

cursorColor (class **Foreground)**
Specify the color to use for the text cursor. The default is **black**.

eightBitInput (class **EightBitInput**)
If **true**, **xterm** presents a metacharacter typed on the keyboard as single character with the eighth bit turned on. If **false**, it converts metacharacters into a two-character sequence with the character itself preceded by **<esc>**. The default is **true**.

eightBitOutput (class **EightBitOutput**)
Specify whether eight-bit characters sent from the host should be accepted as is, or stripped when printed. The default is **true**.

font (class **Font)**
Name the normal font. The default font is **fixed**.

font1 (class **Font1)**
Name the first alternative font.

font2 (class **Font2)**
Name the second alternative font.

font3 (class **Font3)**
Name the third alternative font.

font4 (class **Font4)**
Name the fourth alternative font.

font5 (class **Font5)**
Name the fifth alternative font.

font6 (class **Font6)**
Name the sixth alternative font.

geometry (class **Geometry)**
Set the preferred size and position of the VT102 window.

internalBorder (class **BorderWidth)**
Give the number of pixels between the characters and the window's border. The default is two.

jumpScroll (class **JumpScroll)**
Specify whether to use jump scrolling. The default is **true**.

logFile (class **Logfile)**
Name the file into which a terminal session is logged. The default is **XtermLog.***XXXXX*, where *XXXXX* is the process identifier of **xterm**).

logging (class **Logging)**
Specify whether to log a terminal session. The default is **false**.

logInhibit (class **LogInhibit)**
Specify whether to inhibit the logging of a terminal session. The default is **false**.

loginShell (class **LoginShell)**
Specify whether the shell to be run in the window should be started as a login shell. The default is **false**.

marginBell (class **MarginBell)**
Specify whether to ring the bell when the user's typing approaches the right margin. The default is **false**.

multiClickTime (class **MultiClickTime)**
Set the maximum time, in milliseconds, between multi-click select events. The default is 250 milliseconds.

multiScroll (class **MultiScroll)**
Specify whether to scroll asynchronously. The default is **false**.

nMarginBell (class **Column)**
If the margin bell is enabled, set the point at which **xterm** rings it, in number of characters to the left of the right margin.

pointerColor (class **Foreground)**
Set the foreground color of the mouse cursor. The default is **XtDefaultForeground**.

pointerColorBackground (class **Background)**
Set the background color of the mouse cursor. The default is **XtDefaultBackground**.

pointerShape (class **Cursor)**
Name the shape of the mouse cursor. The default is **xterm**.

resizeGravity (class **ResizeGravity)**
Dictate the behavior when the user changes the height of the window. **NorthWest** specifies that the top line of text on the screen stay fixed. If the window is made shorter, drop lines from the bottom; if the window is made taller, add blank lines at the bottom. This is compatible with the behavior in **R4**. **SouthWest** (the default) specifies that the bottom line of text on the screen stay fixed. If the window is made taller, scroll down additional saved lines onto the screen; if the window is made shorter, scroll lines off the top of the screen and drop the top saved lines.

reverseVideo (class **ReverseVideo)**
Specify whether to simulate reverse video. The default is **false**.

reverseWrap (class **ReverseWrap)**
Specify whether to enable reverse-wraparound. The default is **false**.

saveLines (class **SaveLines)**
Set the number of lines to save beyond the top of the screen. The default is 64.

scrollBar (class **ScrollBar)**
Specify whether to display the scrollbar should be displayed. The default is **false**.

scrollTtyOutput (class **ScrollCond)**
Specify whether output to the terminal automatically moves the scrollbar to the bottom of the scrolling region. The default is **true**.

scrollKey (class **ScrollCond)**
Specify whether pressing a key automatically moves the scrollbar to the bottom of the scrolling region. The default is **false**.

scrollLines (class **ScrollLines)**
Set the number of lines that the actions **scroll-back** and **scroll-forward** should use as a default. The default is one.

signalInhibit (class **SignalInhibit)**
Specify whether to disallow the entries in the menu **Main Options** that send signals to **xterm**. The default is **false**.

tekGeometry (class **Geometry)**
Set the preferred size and position of the Tektronix window.

tekInhibit (class **TekInhibit)**
Specify whether to disallow Tektronix mode. The default is **false**.

tekSmall (class **TekSmall)**
Specify whether to start the Tektronix-mode window in its smallest size if no explicit geometry is given. This is useful when running **xterm** on displays with small screens. The default is **false**.

tekStartup (class **TekStartup)**
Specify whether **xterm** should begin in Tektronix mode. The default is **false**.

titeInhibit (class **TiteInhibit)**
Specify whether **xterm** should remove the entries **ti** and **te** (which switch between alternate screens on startup of many screen-oriented programs) from the environmental variable **TERMCAP**. If set, **xterm** also ignores the escape sequence to switch to the alternate screen.

translations (class **Translations)**
Specify the key and button bindings for menus, selections, "programmed strings," etc. For details, see the section on actions, below.

visualBell (class **VisualBell)**
Specify whether to use a visible bell (i.e., flashing) instead of an audible bell when **<ctrl-G>** is received. The default is **false**.

tek4014 Resources

The following resources are specified as part of the **tek4014** widget (class **Tek4014**):

width (class **Width)**
Set the width of the Tektronix window, in pixels.

height (class **Height)**
Set the height of the Tektronix window, in pixels.

fontLarge (class **Font)**
Name the large font to use in the Tektronix window.

font2 (class **Font)**
Name font number 2 to use in the Tektronix window.

font3 (class **Font)**
Name font number 3 to use in the Tektronix window.

fontSmall (class **Font)**
Name the small font to use in the Tektronix window.

initialFont (class **InitialFont)**
Name which of the four Tektronix fonts to use initially. Values are the same as for the **set-tek-text** action. The default is **large**.

ginTerminator (class **GinTerminator)**
Specify what character (or characters) should follow a **GIN** report or status report. The possibilities are as follows:

none	Send no terminating characters
CRonly	Send CR
CR&EOT	Sends both CR and EOT.

The default is **none**.

The documentation for the Athena **SimpleMenu** widget describes the resources that may be specified for the various menus. The following lists the name and classes of the entries in each menu.

Main Menu

xterm's main menu is invoked by holding down the **<ctrl>** key and pressing the left mouse button. It displays the contents of resource **mainMenu**, and has the following entries:

Secure Keyboard
Invokes the resource **securekbd (**class **SmeBSB)**, which triggers the action **secure()**.

Allow SendEvents
Invokes the resource **allowsends (**class **SmeBSB)**, triggers the action **allow-send-events()**.

Log to File
Invoke the resource **logging (**class **SmeBSB)**, which triggers the action **set-logging()**. This turns on logging for this window.

Redraw Window
Invoke the resource **redraw (**class **SmeBSB)**, which triggers the action **redraw()**.

Send Stop Signal
Invoke the resource **suspend (**class **SmeBSB)**, which triggers the action **send-signal()** to support job control.

Send Cont Signal
Invoke the resource **continue (**class **SmeBSB)**, which triggers the action **send-signal()** on systems that support job control.

Send INT Signal
Invoke the resource **interrupt (**class **SmeBSB)**, which triggers the action **send-signal()**. This sends an interrupt signal to **xterm**.

Send HUP Signal
Invoke the resource **hangup (**class **SmeBSB)**, which triggers the action **send-signal()**. This sends a HUP (hang-up) signal to **xterm**.

Send TERM Signal
Invoke the resource **terminate (**class **SmeBSB)**, which triggers the action **send-signal()**. This sends a TERM (terminate) signal to **xterm**.

Send KILL Signal
Invoke the resource **kill (**class **SmeBSB)**, which triggers the action **send-signal()**. This sends a KILL signal to **xterm**.

Quit Invoke the resource **quit (**class **SmeBSB)**, which triggers the action **quit()**.

Menu VT Options

xterm's menu **VT Options** is invoked by holding down the **<ctrl>** key, and pressing the center mouse button (or, if you have a two-button mouse, pressing both mouse buttons simultaneously). It displays the contents of resource **vtOptions** and has the following entries:

Enable Scrollbar
This invokes resource **scrollbar (**class **SmeBSB)**, which triggers the action **set-scrollbar()**. This toggles displaying the scrollbar.

Enable Jump Scroll
This invokes resource **jumpscroll (**class **SmeBSB)**, which triggers the action **set-jumpscroll()**. This toggles jump-scrolling.

Enable Reverse Video
This invokes resource **reversevideo (**class **SmeBSB)**, which triggers the action **set-reverse-video()**. This toggles putting the screen into reverse video.

Enable Auto Wraparound
This invokes resource **autowrap (**class **SmeBSB)**, which triggers the action **set-autowrap()**. This toggles auto-wraparound.

Enable Reverse Wraparound
This invokes resource **reversewrap (**class **SmeBSB)**, which triggers the action **set-reversewrap()**. This toggles reverse-wraparound.

Enable Auto Linefeed
This invokes resource **autolinefeed (**class **SmeBSB)**, which triggers the action **set-autolinefeed()**.

Enable Application Cursor Keys
This invokes resource **appcursor (**class **SmeBSB)**, which triggers the action **set-appcursor()**.

Enable Application Keypad
This invokes resource **appkeypad (**class **SmeBSB)**, which triggers the action **set-appkeypad()**.

Scroll to Bottom on Keypress
This invokes resource **scrollkey (**class **SmeBSB)**, which triggers the action **set-scroll-on-key()**.

Scroll to Bottom on Tty Output
This invokes resource **scrollttyoutput (**class **SmeBSB)**, which triggers the action **set-scroll-on-tty-output()**.

All 80/132 Column Switching
This invokes resource **allow132 (**class **SmeBSB)**, which triggers the action **set-allow132()**.

Enable Curses Emulation
This invokes resource **cursesemul (**class **SmeBSB)**, which triggers the action **set-cursesemul()**.

Enable Visual Bell
This invokes resource **visualbell (**class **SmeBSB)**, which triggers the action **set-visualbell()**.

Enable Margin Bell
This invokes resource **marginbell (**class **SmeBSB)**, which triggers the action **set-marginbell()**.

Show Alternate Screen
This invokes resource **altscreen (**class **SmeBSB)**, which shows the alternate screen. This entry currently is disabled.

Do Soft Reset
This invokes resource **softreset (**class **SmeBSB)**, which triggers the action **soft-reset()**.

Do Full Reset
This invokes resource **hardreset (**class **SmeBSB)**, which triggers the action **hard-reset()**.

Reset and Clear Saved Lines
This invokes resource **clearsavedlines (**class **SmeBSB)**, which triggers the action **clear-saved-lines()**.

Show Tek Window
This invokes resource **tekshow (**class **SmeBSB)**, which triggers the action **set-visibility()**.

Switch to Tek Mode
This invokes resource **tekmode (**class **SmeBSB)**, which triggers the action **set-terminal-type()**. This switches the terminal to Tektronix emulation.

Hide VT Window
This invokes **vthide (**class **SmeBSB)**, which triggers the action **set-visibility()**.

Menu VT Fonts

xterm's menu **VT Fonts** is invoked by holding down the **<ctrl>** key, and pressing the right mouse button. It displays the contents of resource **fontMenu**, and has the following entries:

Default
Invoke the resource **fontdefault (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets the default font.

Unreadable
Invoke the resource **font1 (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets alternate font 1, which is an invisible font.

Tiny This invokes resource **font2 (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets alternate font 2, which is the tiny font.

Small This invokes resource **font3 (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets alternate font 3, which is the small font.

Medium
This invokes resource **font4 (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets alternate font 4, which is the medium font.

Large **This invokes resource font5 (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets alternate font 5, which is the large font.

Huge This invokes resource **font6 (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This alternative font 6, which is the huge font.

Escape Sequence
This invokes resource **fontescape (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets the escape sequence for switching fonts. By default, this option is not enabled.

Selection
This invokes resource **fontsel (**class **SmeBSB)**, which triggers the action **set-vt-font()**. This sets a new font.

tekMenu Resources

tekMenu has the following entries:

tektextlarge (class **SmeBSB)**
Invoke the action **set-tek-text()**.

tektext2 (class **SmeBSB)**
Invoke the action **set-tek-text()**.

tektext3 (class **SmeBSB)**
Invoke the action **set-tek-text()**.

tektextsmall (class **SmeBSB)**
Invoke the action **set-tek-text()**.

line1 (class **SmeLine)**
This is a separator.

tekpage (class **SmeBSB)**
Invoke the action **tek-page()**.

tekreset (class **SmeBSB)**
Invoke the action **tek-reset()**.

tekcopy (class **SmeBSB)**
Invoke the action **tek-copy()**.

line2 (class **SmeLine)**
This is a separator.

vtshow (class **SmeBSB)**
Invoke the action **set-visibility()**.

vtmode (class **SmeBSB)**
Invoke the action **set-terminal-type()**.

tekhide (class **SmeBSB)**
Invoke the action **set-visibility()**.

Athena Scrollbar Resources

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness)**
Set the width of the scrollbar, in pixels.

background (class **Background)**
Set the color of the scrollbar's background.

foreground (class **Foreground)**
Set the color of the scrollbar's foreground. The "thumb" of the scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

Using the Mouse

Once the VT102 window is created, **xterm** lets you cut text and paste it within the same or other windows.

The selection functions are invoked when the mouse buttons are used with no modifiers, and when they are used with the **<shift>** key. (If you wish, you can use the X utility **xrdb** to change what functions are linked to which keys and mouse buttons. see the section entitled **Actions**, below.)

The left mouse button copies text into the X server's cut buffer. Move the cursor to beginning of the text, and then drag the mouse to the end of the region and release the button. The selected text is highlighted, and is saved in the global cut buffer (i.e., the property **PRIMARY**).

Double-clicking cuts a word of text. Triple-clicking cuts a line of text. Multiple-click is determined by the time from button-up to button-down, so you can change the unit to be selected while you are selecting. If the key/button bindings specify that an X selection is to be made, **xterm** leaves the selected text highlighted for as

long as it owns the selection — that is, until you make another selection, or click the right-mouse button to clear the selection.

The center-mouse button pastes the text from the primary selection, if any; otherwise, it pastes the contents of the cut buffer. Text being pasted is processed as though it were typed from the keyboard.

The right-mouse button extends the current selection. By pressing this button and dragging the mouse, you can "sweep" out a mass of text to cut. **xterm** displays all cut text in reverse video, so you can see exactly what you have selected. If you press the right mouse button while the mouse cursor is closer to the right edge of the selection than the left, **xterm** extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, **xterm** assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection-unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting "hunks" of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Because the cut buffer is globally shared among different applications, you should regard it as a "file" whose contents you know. The terminal emulator and other text programs should treat it as if it were a text file, i.e., the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

If you click the left mouse button when the pointer is in the scroll region, **xterm** moves the adjacent line to the top of the display window.

If you click the right mouse button, **xterm** drops the top line of the display window to the mouse cursor's position.

If you click the middle mouse button, **xterm** moves the display to a position in the saved text that corresponds to the mouse cursor's position in the scrollbar.

Unlike the VT102 window, the Tektronix window does not allow you to copy text. It does allow Tektronix **GIN** mode, and in this mode the mouse cursor changes from an arrow to a cross. Pressing any key sends that key and the current coordinate of the cross cursor. Pressing the left, center, or right mouse buttons return the letters 'l', 'm', and 'r', respectively. If the **<shift>** key is pressed when a pointer button is pressed, the corresponding upper-case letter is sent. To distinguish a mouse button from a key, the high bit of the character is set (but this bit is normally stripped unless the terminal mode is **RAW**; for details, see the entry in the COHERENT Lexicon for **stty**).

Menus

xterm has four menus, named **mainMenu**, **vtMenu**, **fontMenu**, and **tekMenu**. Each menu pops up under the correct combinations of keystrokes and mouse-button motions. Most menus are divided into two sections, separated by a horizontal line. The upper sections contains various modes that can be altered. A check mark appears next to a mode that is currently active. Clicking one of these modes toggles its state. The lower section of the menu are command entries; selecting one of these performs the indicated function.

An **xterm** menu pops up when you press the **<ctrl>** key and the left mouse button in a window. The **MainMenu** contains items that apply to both the VT102 and Tektronix windows. Use the **Secure Keyboard** mode when typing in passwords or other sensitive data in an unsecure environment; for details, see the section on security, below. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate**, and **Kill**. These send, respectively, the signals **SIGCONT**, **SIGTSTP**, **SIGINT**, **SIGHUP**, **SIGTERM**, and **SIGKILL** to the process group of the process running under **xterm** (usually the shell). The function **Continue** is especially useful if the user has accidentally typed **<ctrl-Z>**, suspending the process.

The **vtMenu** sets various modes in the VT102 emulation, and is popped up when the **<ctrl>** key and right mouse button are pressed in the VT102 window. In the command section of this menu, the soft-reset entry resets the scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full-reset entry clears the screen, resets tabs to every eight columns, and resets the terminal modes (such as wrap and smooth scroll) to their initial states just after **xterm** has finished processing the command-line options.

The **fontMenu** sets the font used in the VT102 window. In addition to the default font and a number of alternatives that are set with resources, the menu offers the font last specified by the escape sequence **Set Font** and the current selection as a font name (if **xterm** owns the primary selection).

The **tekMenu** sets various modes in the Tektronix emulation. It pops up when you press the **<ctrl>** key and the

middle mouse button in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

Security

X environments differ in their security consciousness. MIT servers, run under **xdm**, can use a "magic cookie" authorization scheme that can provide a reasonable level of security for many people. If your server is only using a host-based mechanism to control access to the server (as described in the Lexicon entry for **xhost**), then if you enable access for a host and other users are also permitted to run clients on that same host, there is every possibility that someone can run an application that will use the basic services of the X protocol to snoop on your activities, possibly even capturing a transcript of everything you type at the keyboard. This is of particular concern when you want to type in a password or other sensitive data. The best solution to this problem is to use a better authorization mechanism than host-based control.

xterm has a a simple mechanism to protect keyboard input. The **xterm** menu (as described above) contains a **Secure Keyboard** entry that, when enabled, uses the **GrabKeyboard** protocol request to ensures that all keyboard input is directed *only* to **xterm**. When an application prompts you for a password (or other sensitive data), you can use the menu to enable **Secure Keyboard**, type in the data, and then use the menu to disable **Secure Keyboard**.

Only one X client at a time can secure the keyboard, so when you attempt to enable **Secure Keyboard**, it may fail. In this case, **xterm** sounds the bell. If the **Secure Keyboard** succeeds, the foreground and background colors will be exchanged (as if you selected the **Reverse Video** entry in the **Modes** menu); they will be exchanged again when you exit secure mode. If the colors do *not* switch, then you should be *very* suspicious that you are being "spoofed." If the application you are running displays a prompt before asking for the password, it is safest to enter secure mode *before* the prompt is displayed, and to make sure that the prompt is displayed correctly (in the new colors), to minimize the probability of spoofing. You can also bring up the menu again and make sure that a check mark appears next to the entry.

xterm automatically disables **Secure Keyboard** mode if your **xterm** window becomes iconified (or otherwise unmapped), or if you start up a reparenting window manager (that places a title bar or other decoration around the window) while in **Secure Keyboard** mode. (This is a feature of the X protocol not easily overcome.) When this happens, **xterm** switches back the foreground and background colors and sounds the bell.

Character Classes

Clicking the middle mouse button twice rapidly selects all characters of the same class (e.g., letters, white space, punctuation) to be selected. Because different people have different preferences for what should be selected (for example, should file names be selected as a whole or only the separate subnames), you can override the default mapping through the use of the resource **charClass** (class **CharClass)**.

This resource consists simply of a list of *range***:***value* pairs, where the *range* is either a single number or *low*-*high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The following gives the default table:

```
static int charClass[128] = {
/* NUL  SOH  STX  ETX  EOT  ENQ  ACK  BEL */
    32,   1,   1,   1,   1,   1,   1,   1,
/*  BS   HT   NL   VT   NP   CR   SO   SI */
     1,  32,   1,   1,   1,   1,   1,   1,

/* DLE  DC1  DC2  DC3  DC4  NAK  SYN  ETB */
     1,   1,   1,   1,   1,   1,   1,   1,
/* CAN   EM  SUB  ESC   FS   GS   RS   US */
     1,   1,   1,   1,   1,   1,   1,   1,

/*  SP    !    "    #    $    %    &    ' */
    32,  33,  34,  35,  36,  37,  38,  39,
/*   (    )    *    +    ,    -    .    / */
    40,  41,  42,  43,  44,  45,  46,  47,

/*   0    1    2    3    4    5    6    7 */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   8    9    :    ;    <    =    >    ? */
    48,  48,  58,  59,  60,  61,  62,  63,
```

```
/*   @    A    B    C    D    E    F    G */
    64,  48,  48,  48,  48,  48,  48,  48,
/*   H    I    J    K    L    M    N    O */
    48,  48,  48,  48,  48,  48,  48,  48,

/*   P    Q    R    S    T    U    V    W */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   X    Y    Z    [    \    ]    ^    _ */
    48,  48,  48,  91,  92,  93,  94,  48,

/*   ‘    a    b    c    d    e    f    g */
    96,  48,  48,  48,  48,  48,  48,  48,
/*   h    i    j    k    l    m    n    o */
    48,  48,  48,  48,  48,  48,  48,  48,

/*   p    q    r    s    t    u    v    w */
    48,  48,  48,  48,  48,  48,  48,  48,
/*   x    y    z    {    |    }    ~  DEL */
    48,  48,  48, 123, 124, 125, 126,   1};
```

For example, the string

```
33:48,37:48,45-47:48,64:48
```

tells **xterm** to treat the exclamation mark, percent sign, dash, period, slash, and ampersand characters as characters and numbers. This is very useful for cutting and pasting e-mail addresses and file names.

Actions

You can rebind keys (or sequences of keys) to arbitrary strings for input by changing the translations for the **vt100** or **tek4014** widgets. Changing the translations for events other than key and button events is not expected, and can cause unpredictable behavior. The following actions are provided for using within the **vt100** or **tek4014 translations** resources:

bell([*percent***])**
Ring the keyboard bell at the specified percentage above or below the base volume.

ignore()
Ignore the event, but check for special cursor-position escape sequences.

insert() Insert the character or string associated with the key that was pressed.

insert-seven-bit()
A synonym for **insert()**

insert-eight-bit()
Insert an eight-bit (meta) version of the character or string associated with the key that was pressed. The exact action depends on the value of the resource **eightBitInput**.

insert-selection(*sourcename***[, ...])**
Insert the string found in the selection or cut-buffer *sourcename* **xterm** checks the sources in the order given (case is significant) until it finds one. Commonly-used selections include **PRIMARY**, **SECONDARY**, and **CLIPBOARD**. Cut buffers are typically named **CUT_BUFFER0** through **CUT_BUFFER7**.

keymap(*name***)**
Dynamically define a new translation table whose resource name is *name* with the suffix **Keymap** (case is significant). The name **None** restores the original translation table.

popup-menu(*menuname***)**
Display the specified pop-up menu. Valid names include **mainMenu**, **vtMenu**, **fontMenu**, and **tekMenu** Case is significant.

secure()
Toggle the **Secure Keyboard** mode described in the section on section, above. It is invoked from the entry **securekbd** entry in menu **mainMenu**.

select-start()
Begins text selection at the current location of the mouse cursor. For details on making selection, see the section above entitled **Using the Mouse**.

select-extend()
Track the pointer and extend the selection. It should only be bound to **Motion** events.

select-end(*destname*[, ...])
Puts the currently selected text into all of the selections or cut buffers specified by *destname*.

select-cursor-start()
Like **select-start**, except that it begins the selection at the current position of the text cursor.

select-cursor-end(*destname*[, ...])
Like **select-end**, except that you should use it with **select-cursor-start**.

set-vt-font(*d/1/2/3/4/5/6/e/s*[,*normalfont*[, *boldfont*]])
Set the font or fonts being used in the VT102 window. The first argument is a character that specifies the font to be used: **d** or **D** indicate the default font (the font initially used when **xterm** was started); **1** through **6** indicate the fonts specified by the resources **font1** through **font6; e** or **E** indicate the normal and bold fonts that have been set through escape codes (or specified as the second and third action arguments, respectively); and **s** or **S** indicate the font selection (as made by programs such as **xfontsel**) indicated by the second action argument.

start-extend()
Like **select-start**, except that the selection is extended to the location of the mouse cursor.

start-cursor-extend()
Like **select-extend**, except that the selection is extended to the current position of the text cursor.

string(*string*)
Insert the specified text string as if it had been typed. Quotation is necessary if the string contains white space or non-alphanumeric characters. If the string argument begins with the characters '0x', **xterm** interprets it as a hexadecimal character constant.

scroll-back(*count*[,*units*])
Scroll the text window backward, so that text that had previously scrolled off the top of the screen is now visible. *count* gives the number of *units* (which may be **page**, **halfpage**, **pixel**, or **line**) by which to scroll.

scroll-forw(*count*[,*units*])
Like **scroll-back**, except that it scrolls the other direction.

allow-send-events(*on/off/toggle*)
Set or toggle the **allowSendEvents** resource. This feature is also invoked by the entry **allowsends** in **mainMenu**.

set-logging(*on/off/toggle*)
Set or toggle the **logging** resource. This, too, is invoked by the **logging** entry in **mainMenu**.

redraw()
Redraw the window. It is invoked by the entry **redraw** in **mainMenu**.

send-signal(*signame*)
Send the signal *signame* to the **xterm** subprocess (the shell or program specified with the **-e** command-line option). It is invoked by the entries **suspend**, **continue**, **interrupt**, **hangup**, **terminate**, and **kill** in **mainMenu**. Allowable signal names are **tstp** (if supported by the operating system), **suspend** (same as **tstp**) **cont** (if supported by the operating system), **int**, **hup**, **term**, **quit**, **alrm**, **alarm** (same as **alrm**), and **kill**. Case is not significant.

quit() Send the signal **SIGHUP** to the subprogram, and exit from **xterm**. It is also invoked by the **quit** entry in **mainMenu**.

set-scrollbar(*on/off/toggle*)
Toggles the **scrollbar** resource. It is invoked by the **scrollbar** entry in **vtMenu**.

set-jumpscroll(*on/off/toggle*)
Toggle the resource **jumpscroll**. It is also invoked by the entry **jumpscroll** in **vtMenu**.

set-reverse-video(*on/off/toggle*)
Toggle the resource **reverseVideo**. It is also invoked by entry **reversevideo** entry in **vtMenu**.

set-autowrap(*on/off/toggle***)**
Toggle automatic wrapping of long lines. It is invoked by entry **autowrap** in **vtMenu**.

set-reversewrap(*on/off/toggle***)**
Toggle the resource **reverseWrap**. It is invoked by by the entry **reversewrap** in **vtMenu**.

set-autolinefeed(*on/off/toggle***)**
Toggle automatic insertion of linefeeds. It is invoked by the entry **autolinefeed** in **vtMenu**.

set-appcursor(*on/off/toggle***)**
Toggle the handling of Application Cursor Key mode. It is invoked by the entry **appcursor** in **vtMenu**.

set-appkeypad(*on/off/toggle***)**
Toggle the handling of Application Keypad mode. It is invoked by the entry **appkeypad** in **vtMenu**.

set-scroll-on-key(*on/off/toggle***)**
Toggle the resource **scrollKey**. It is invoked from the entry **scrollkey** in **vtMenu**

set-scroll-on-tty-output(*on/off/toggle***)**
Toggle the resource **scrollTtyOutput.** It is invoked from the entry **scrollttyoutput** in **vtMenu**.

set-allow132(*on/off/toggle***)**
Toggle the resource **c132**. It is invoked from the entry **allow132** in **vtMenu**.

set-cursesemul(*on/off/toggle***)**
Toggle the resource **curses**. It is invoked from the entry **cursesemul** in **vtMenu**.

set-visual-bell(*on/off/toggle***)**
Toggle the resource **visualBell**. It is invoked entry **visualbell** in **vtMenu**.

set-marginbell(*on/off/toggle***)**
Toggle the resource **marginBell**. It is invoked from entry **marginbell** in **vtMenu**.

set-altscreen(*on/off/toggle***)**
Toggle between alternative and main screens.

soft-reset()
Reset the scrolling region, tabs, window size, and cursor keys.

hard-reset()
Resets the scrolling region, tabs, window size, and cursor keys, and clears the screen. It is invoked from the entry **hardreset** in **vtMenu**.

clear-saved-lines()
Perform **hard-reset()** and clear the history of lines saved from the top of the screen. It is invoked from the entry **clearsavedlines** in **vtMenu**.

set-terminal-type(*type***)**
Set the terminal type to *type*, which must be either **vt** or **tek**. It is invoked by the entry **tekmode** in **vtMenu** and the entry **vtmode** in **tekMenu**.

set-visibility(*vt/tek,on/off/toggle***)**
Controls whether the **vt** or **tek** windows are visible. It is invoked from the entries **tekshow** and **vthide** in **vtMenu**, and the entries **vtshow** and **tekhide** in **tekMenu**.

set-tek-text(*large/2/3/small***)**
Set font used in the Tektronix window to the value of the resources **tektextlarge**, **tektext2**, **tektext3**, and **tektextsmall**, according to the argument. It is also by the entries of the same names as the resources in **tekMenu**.

tek-page()
Clears the Tektronix window. This is invoked by the entry **tekpage** in **tekMenu**.

tek-reset()
Resets the Tektronix window. It is invoked by the entry **tekreset** in **tekMenu**.

tek-copy()
Copy the escape codes used to generate the current window's contents to a file in the current directory whose name begins with the string **COPY**. It is invoked from the entry **tekcopy** in **tekMenu**.

visual-bell()
Flash the window quickly.

The Tektronix window also has the following action:

gin-press(*l/L/m/M/r/R***)**
Send the indicated graphics-input code.

Resources

The following gives the default settings for the resources used in the VT102 window:

```
Shift <KeyPress> Prior:       scroll-back(1,halfpage) \n\
Shift <KeyPress> Next:        scroll-forw(1,halfpage) \n\
Shift <KeyPress> Select:      select-cursor-start() \
                              select-cursor-end(PRIMARY, CUT_BUFFER0) \n\
Shift <KeyPress> Insert:      insert-selection(PRIMARY, CUT_BUFFER0) \n\
        ~Meta<KeyPress>:      insert-seven-bit() \n\
         Meta<KeyPress>:      insert-eight-bit() \n\
       !Ctrl <Btn1Down>:      popup-menu(mainMenu) \n\
  !Lock Ctrl <Btn1Down>:      popup-menu(mainMenu) \n\
      ~Meta <Btn1Down>:       select-start() \n\
    ~Meta <Btn1Motion>:       select-extend() \n\
       !Ctrl <Btn2Down>:      popup-menu(vtMenu) \n\
  !Lock Ctrl <Btn2Down>:      popup-menu(vtMenu) \n\
 ~Ctrl ~Meta <Btn2Down>:      ignore() \n\
   ~Ctrl ~Meta <Btn2Up>:      insert-selection(PRIMARY, CUT_BUFFER0) \n\
       !Ctrl <Btn3Down>:      popup-menu(fontMenu) \n\
  !Lock Ctrl <Btn3Down>:      popup-menu(fontMenu) \n\
 ~Ctrl ~Meta <Btn3Down>:      start-extend() \n\
    ~Meta <Btn3Motion>:       select-extend() \n\
                                        <BtnUp>:select-end(PRIMARY, CUT_BUFFER0) \n\
              <BtnDown>:      bell(0)
```

The following gives the default settings for the resources used in the Tektronix window:

```
      ~Meta<KeyPress>:         insert-seven-bit() \n\
       Meta<KeyPress>:         insert-eight-bit() \n\
     !Ctrl <Btn1Down>:         popup-menu(mainMenu) \n\
!Lock Ctrl <Btn1Down>:         popup-menu(mainMenu) \n\
     !Ctrl <Btn2Down>:         popup-menu(tekMenu) \n\
!Lock Ctrl <Btn2Down>:         popup-menu(tekMenu) \n\
Shift ~Meta<Btn1Down>:        gin-press(L) \n\
      ~Meta<Btn1Down>:        gin-press(l) \n\
Shift ~Meta<Btn2Down>:        gin-press(M) \n\
      ~Meta<Btn2Down>:        gin-press(m) \n\
Shift ~Meta<Btn3Down>:        gin-press(R) \n\
      ~Meta<Btn3Down>:        gin-press(r)
```

Below is a sample how of you can use the **keymap()** action to add special keys for entering commonly-typed works:

```
*VT100.Translations: #override <Key>F13: keymap(dbx)
*VT100.dbxKeymap.translations: \
       <Key>F14:      keymap(None) \n\
       <Key>F17:      string("next") string(0x0d) \n\
       <Key>F18:      string("step") string(0x0d) \n\
       <Key>F19:      string("continue") string(0x0d) \n\
       <Key>F20:      string("print ") insert-selection(PRIMARY, CUT_BUFFER0)
```

Environment

xterm sets the environment variables **TERM**. It also uses and sets the environment variable **DISPLAY** to specify which bit-mapped display terminal to use. It sets the environment variable **WINDOWID** to the X-Window identification number of the **xterm** window.

See Also

pty, resize X clients, xvt
COHERENT Lexicon: **curses, pty, termcap, terminfo, utmp, vi**

LEXICON

Notes

The following example, by Ernest Cline (cline@usceast.cs.scarolina.edu), shows how to re-configure **xterm** to obtain a character with a value above 127. It displays an accented 'A' if you press (F1) and then 'A':

```
*VT100.GraveKeymap.translations: \
      Shift <Key>A: string(0xc0) keymap(None) keymap(Latin1)
```

Large pastes do not work on some systems. This is not a bug in **xterm**; rather, it is a bug in the pseudo-terminal driver of those systems. **xterm** feeds large pastes to the pseudo-terminal only as fast as the driver **pty** can accept data, but some **pty** drivers do not return enough information to know if the write has succeeded.

xterm was written by many people, including Loretta Guarino Reid, Joel McCormack, Bob McNamara, and Terry Weissman of Digital Equipment Corporation; Edward Moy of the University California, Berkeley; Ralph R. Swick, Mark Vandevoorte, Jim Gettys, Ron Newman, and Jonathan Kamens of MIT Project Athena; Jim Fulton and Bob Scheifler of the MIT X Consortium; Doug Mink of SAO; Steve Pitschke of Stellar; and Dave Serisky of Hewlett-Packard.

xtetris — X Client

Wildly amusing implementation of Tetris
xtetris [*toolkitoption ...* **]**

xtetris is an X Windows implementation of the popular game Tetris. This game drops a series of blocks from the top of a column. Each block is randomly selected from the set of six possible shapes. The point of the game is to maneuver the blocks as they fall, and so build a solid mass at the bottom of the column.

As a block drops, you can move it to the left or to the right by clicking the left or right mouse buttons, respectively. Pressing the shift key while clicking the left mouse button rotates the falling shape counter-clockwise; pressing the shift key while clicking the right mouse button rotates it clockwise. Pressing the middle mouse button drops the shape quickly.

You can also use the keyboard:

h Move the block to the left.

l Move the block to the right.

j Rotate clockwise.

k Rotate counter-clockwise.

<space>
Drop quickly.

You can also use arrow keys, as follows:

(æ) Move left.

(Æ) Move right.

(°) Rotate clockwise.

(ª) Rotate counter-clockwise.

You score points for each block that comes to rest on the gradually accumulating pile of blocks. Different blocks in different orientations have different point values. If you complete a row of blocks across the column, that row is removed and the entire pile drops by the width of the row, thus buying you more time to play. The longer you play, the faster the blocks fall. The game ends when the pile reaches the top of the screen and no more blocks can fall.

xtetris keeps a table of high scores. You can avoid recording your score by using the option **-noscore**.

xtetris requires the defaults file **/usr/X11/lib/app-defaults/Xtetris**. You can configure almost everything about **xtetris** in its defaults file. **xtetris** comes with two sample defaults files: **Xtetris.c** and **Xtetris.bw**. As their names imply, the former is for color systems, and the latter is for monochrome. Some objects are unsightly, or disappear entirely, if you run **xtetris** with the wrong defaults file; therefore, you should link **Xtetris** to the appropriate file.

Options

xtetris recognizes the following command-line options:

-bd *color* Set the color of the border to *color*.

-bg *color* Set the color of the background to *color*.

-boxsize *boxsize*
Set the width of the square blocks that comprise the falling objects. The overall size of the game board adjusts to *boxsize*.

-bw Use black-and-white defaults file.

-fg *color* Set the color of the foreground to *color*.

-fn *font* Use *font* in the display.

-geometry *geometry*
Set the geometry of the program's window to *geometry*. The term "geometry" means the dimensions of the window and its location on the screen. *geometry* has the form *width*×*height*±*xoffset*±*yoffset*.

-noscore Do not record your score or show you the score file when you have finished the game.

-rv Simulate reverse video by exchanging the foreground and background colors.

-score Runs **xtetris** using the scorefile, if it exists.

-speed *speed*
Sets the game's speed. By default, *speed* is ten. A *speed* of 20 doubles game speed; a *speed* of five makes it half as fast. **-speed 50** makes for an extremely fast game; however, you will need a fast processor to support it. Keep in mind that as you knock out rows, the game's speed increases. If you set your speed below the standard of ten, your score will not be recorded in the score file.

-xrm *resourcestring*
Use *resourcestring* to define a resource.

Files

/usr/lib/X11/app-defaults/Xtetris — Default defaults file
/usr/lib/X11/app-defaults/Xtetris.c — Color defaults file
/usr/lib/X11/app-defaults/Xtetris.bw — Monochrome defaults file
/usr/lib/X11/lib/tetris_scores — Top ten high scores

Environmental Variables

xtetris reads the environmental variable **XTETRIS**, which gives the name of the scores file.

See Also

puzzle, xgas, X clients

Notes

A bug in X11R5 makes the file **/usr/lib/X11/app-defaults/Xtetris** necessary: the pop-up score-box and the information box do not receive a resource in the X server's resource data base.

 xtetris was written by Dan Greening, Didier Tallot, Phill Everson, Martyn Shortley, and Adam Marguilies.

xvt — X Client

VT100 emulator
xvt [*options* **]**

xvt emulates the DEC VT-100 terminal under X. It is a scaled-down version of the X client **xterm**: it eschews some of that client's more esoteric features to improve speed and reduce size. Specifically, **xvt** does not implement the Tektronix 4014 emulation, session logging, and toolkit-style configurability. As a result, **xvt** uses much less memory than does **xterm** — a significant advantage on a machine with limited RAM.

Options

xvt recognizes the following command-line options. With the exception of **-msg**, these are a subset of those supported by **xterm**. Most command-line options have X-resource equivalents; these are listed in the following

table.

-bd *color*
Set the color of the border to *color*. For a list of recognized colors, see the file **/usr/X11/lib/rgb.txt**.

-bg *color*
Set the background of the **xvt** window to *color*.

-bw *number*
Set the width of the window's border to *number* pixels.

-cc *characterclassrange*:*value***[,...]**
Set or modify the characters that determine what is a word when a double click is used to select a word of text. This is identical to **xterm**'s option **-cc**. For details, see the Lexicon entry for **xterm**.

-display *display*
Attempt to open the **xvt** window on *display*. In the absence if this option, **xvt** uses the display specified by the environmental variable **DISPLAY**.

-e *command* **[** *arguments* **]**
Run *command* with its command-line *arguments* in the **xvt** window. If this option is used, it must be the last on the command line. If there is no **-e** option then the default is to run the program specified by environmental variable **SHELL** or failing that, the Bourne shell **sh**. This option also causes the window title and icon name to be set to the name of the program being executed, if the are not overwritten by a more specific option.

-fb *fontname*
Set the font used for the VT-100 bold rendition. Unlike **xterm**, **xvt** will not try to create bold text by displacing and OR'ing an ordinary font. Therefore, if you want bold highlighting to work, you must use this option to specify a suitable bold font.

-fg *color*
Set the foreground of the **xvt** window to *color*.

-font *fontname*
Set the main text font used by **xvt**. For a list of recognized font names, see the files **/usr/X11/lib/fonts/misc/fonts.alias** and **/usr/X11/lib/fonts/75dpi/fonts.alias**.

-geometry *geometry*
Create the window with the specified X-window geometry.

-iconic Start up with **xvt** iconized.

-msg Enable messages to the terminal window from programs like **write**. By default, **xvt** windows have messages disabled. Executing an **xvt** with the **-msg** option has the same effect as running it normally and then executing the command **mesg y** to enable messages.

-n *name*
Set the name **xvt** displays in its icon. This also sets the name that **xvt** shows in its title, unless you use the option **-title**, described below.

-name *name*
Obtain resources under *name* rather than as **xvt**. *name* should not contain the characters '.' or '*'.

-rw Enable reverse wrapping of the text cursor. This lets you edit shell commands that are longer than the screen is wide.

-sb Start up with the scrollbar visible. The scrollbar can be displayed or hidden at any time simply by holding down the **<ctrl>** key and clicking any mouse button. Note that **xvt** saves text regardless of whether the scrollbar is displayed.

-sl *number*
Set the number of lines that **xterm** saves.

-title *name*
-T *name*
Set the name that **xvt** displays in its title bar

Resources

With the exception of the options **-e**, **-display**, and **-name**, every command-line option has an X-resource counterpart. Like **xterm**, **xvt** uses the class name **XTerm** and so resource options set for **XTerm** also work for **xvt**. The following table lists command-line options and their resource counterparts:

Command-line Option	*Instance*	*Class*
-bd	**borderColor**	**BorderColor**
-bg	**background**	**Background**
-bw	**borderWidth**	**BorderWidth**
-cc	**charClass**	**CharClass**
-display	*None*	*None*
-e	*None*	*None*
-fb	**boldFont**	**BoldFont**
-fg	**foreground**	**Foreground**
-font	**font**	**Font**
-geometry	**geometry**	**Geometry**
-iconic	**iconic**	**Iconic**
-msg	**messages**	**Messages**
-n	**iconName**	**IconName**
-name	*None*	*None*
-rw	**reverseWrap**	**ReverseWrap**
-sb	**scrollBar**	**ScrollBar**
-sl	**saveLines**	**SaveLines**
-title	**title**	**Title**

Names, Titles, and Icon Names

One occasionally confusing aspect of **xvt** and other X applications is the collection of names that an application window can have, and the relationship between the names and the command-line options used to set them. This section attempts to make the situation a bit clearer in the case of **xvt**.

Each terminal window has three names: its *resource name*, its *title*, and its *icon name*. These three names are distinct and have different functions, although they usually have the same value. The resource name names the command used to identify X resource options in the resources data base. The title is the text displayed in the title bar, if there is one. The icon name is the name that appears in the window's icon or represents it in the icon manager's window.

The options **-name** and **-e** set both the title and the icon name in addition to their main function. Option **-n** sets the title and the icon name. Conflicts are resolved by giving the options the following priorities: first **-e**, then **-name**, **-n**, and finally ~.BR -title . Therefore, **-e** sets the title only if none of the other options is used.

Scroll Bar

xvt saves the lines of text that scroll off the top of its window, up to a preset maximum. You can view these lines by manipulating the **xvt** window's scrollbar. To display or hide the scrollbar, press the **<ctrl>** key and click any mouse button.

When the scrollbar is displayed, click the left mouse button to roll up a few lines; click the right mouse button to click down a few lines. Assuming that **xvt** has buffered enough lines, the distance scrolled with either button is equal to the number of lines between the cursor and the top of the window. Hence, pressing the left cursor opposite a line of text moves that line to be the top of the window; and pressing the right button moves the top line down so that it is opposite the cursor.

To scroll continually, press move the mouse cursor to the scroll bar, hold down the middle mouse button, and drag it up or down.

Cutting and Pasting Text

xvt, like **xterm**, lets you cut and paste text.

To paste text, move the mouse cursor to the spot where you want to "drop" the text, then press the middle mouse button. (If your mouse has only two buttons, press both buttons simultaneously.) **xvt** will handle the pasted text just as if you had typed it from the keyboard. To drop text into a file, you can first invoke an editor, or use a command like **cat**.

To cut text, press the left mouse button at the point where you want to begin cutting; drag the mouse to the end of the block of text you wish to cut; and release the left mouse button. **xvt** highlights the text you have selected, to

show that it is cut. Cut text is copied into the X server's cut buffer and its primary selection (that is, the property **PRIMARY**).

To select a large block of text, click the left mouse button to begin selection; then use the scrollbar to move lower in the file; and finally click the right mouse button.

Double-clicking the left mouse button cuts the word that is under the mouse cursor, with a "word" defined as a collection of ASCII characters delineated by white space or newline characters. Triple-clicking the left mouse button cuts the whole line. word and a triple click selecting a whole line. For this purpose, a word is a sequence of characters in the same class. The default character classes are: To change the character classes so that, for example, you can select a COHERENT path name or a mail address in one double click, use the option **-cc** or modify the resource **charClass**. You can combine multiple clicking and dragging to select a sequence of consecutive words or lines.

Although **xvt** mimics the behavior of **xterm** in its support of text selection and insertion, there are a couple of minor differences:

- **xvt** respects **<tab>** characters in selected text and does not automatically convert them into spaces as **xterm** does.
- **xvt** lets you abort a text insertion if you realize before you have released the middle mouse button that you have made a mistake.

See Also

X clients, xterm

Notes

In some instances, **xvt** does not permit you to backspace over a previously typed character and erase. To work around this problem, you must issue the command

```
stty erase ^H
```

If you use **ksh**, include this command in the file you name in the environmental variable **ENV**: the Korn shell reads this file every time it spawns a sub-shell, and so will configure your terminal device correctly. For details on how to set and use **ENV**, see the Lexicon entries for **ksh** and **.kshrc**.

xvt was written by John Bovey of the University of Kent. It was ported to COHERENT by Harry Pulley.

xwd — X Client

Dump an image of an X window

xwd [-add *value***] [-debug] [-display** *display***] [-frame] [-help] [-icmap] [-id** *id***] [-name** *name***] [-nobdrs] [-out** *file***] [-root] [-screen] [-xy]**

xwd dumps an X window into a file. The contents can either be printed by the X client **xpr**, or redisplayed by the client **xwud**. (For information on these clients, see their entries in this Lexicon.) You can select the window to dump either interactively (by moving the mouse cursor into the window and clicking any mouse button), or by giving on the command line the window's name or identifier.

xwd recognizes the following command-line options:

-add *value*
Add *value* to every pixel, where **value** is a signed integer.

-display *server*
Connect to *server*.

-frame When dumping a window, include its frame.

-help Print a summary of the command-line options.

-icmap Obtain the window's RGB values by reading the first installed color map. By default, **xwd** obtains these values by reading the window's color map.

-id *id* Dump the window with the identifier *id*.

-name *name*
Dumped the window named *name*, as specified in the property **WM_NAME**.

-nobdrs
When dumping the window, do not include pixels that compose the window's border. This is useful in situations where you wish to use the window's contents as an illustration in a document.

-out *file*
Write the output into *file*. The default is to write to the standard output.

-root Dump the screen's root window. Note that this dumps an image of the virtual screen, not the physical screen; so when you redisplay or print this image, you may get more than you expected.

-screen
Execute the **GetImage** request used to obtain the image on the root window, rather than directly on the specified window. This lets you obtain pieces of other windows that overlap the specified window, and capture menus or other pop-up items that are independent windows but which are superimposed upon the specified window.

-xy Use **XY** format dumping instead of the default **Z** format. This option applies to color displays only.

Environment

xwd reads the environmental variable **DISPLAY** to get the numbers of the default host and display.

Files

XWDFile.h — Define the format of the X Window dump file

See Also

xpr, X clients, xwud

Notes

xwd was written by Tony Della Fera of Digital Equipment Corp. and MIT Project Athena, and William F. Wyatt of the Smithsonian Astrophysical Observatory.

xwininfo — X Utility

Display information about a window
xwininfo [-all] [-bits] [-children] [-display *display***] [-english]**
[-events] [-frame] [-help] [-id *id***] [-int] [-metric] [-name** *name***] [-root] [-shape] [-size] [-stats] [-tree] [-wm]**

xwininfo displays information about a window. The information displayed depends upon the command-line options used. If no options are used, it acts as if the **-stats** were used.

You can select the target window either interactively (by moving the mouse cursor into the desired window and then clicking any mouse button), or by specifying it on the command line with the options **-id** or **-name**. **xwininfo** recognizes the following command-line options:

-all Ask for all possible information.

-bits Display the attributes that pertain to the selected window's raw bits and how the selected window is to be stored. These include the selected window's bit gravity, window gravity, backing-store hint, backing-planes value, backing pixel, and whether it has save-under set.

-children Print the identifiers of the select window's root, parent, and child windows.

-display *server*
Connect to *server*.

-english Display all individual height, width, and X and Y positions in inches (and feet, yards, rods, furlongs, and miles if necessary), as well as number of pixels, based on what the server believes the resolution to be. Geometry specifications that are in the form **+x+y** are not changed. **-metric** and **-english** can be enabled simultaneously.

-events Display the window's event masks. This displays both the event mask of events wanted by some client, and the event mask of events not to propagate.

-frame Include the window manager's frames when selecting a window manually.

-help Print a summary of this command.

-id *id* Print information about the window with the identifier *id*. This is useful when debugging an X application whose window is not mapped to the screen, or when using the mouse might be impossible or interfere with the application.

-int Display all window identifiers as decimal values. The default is to display them as hexadecimal values.

-metric Display all individual height, width, and x and y positions in millimeters, as well as number of pixels, based on what the server believes the resolution to be. Geometry specifications that are in the form **+x+y** are not changed. **-metric** and **-english** can be enabled simultaneously.

-name *name*
Print information about the window with name *name*.

-root Print information about the X server's root window. This is useful when the root window is completely obscured.

-shape Display the window's window and border shape extents.

-size Display the window's sizing hints. These include (for both the normal-size hints and the zoom-size hints) the user-supplied location; the program-supplied location; the user-supplied size; the program-supplied size; the minimum size; the maximum size; the resize increments; and the minimum and maximum aspect ratios.

-stats Display the attributes that pertain to the window's appearance and location. These include the location of the window, its width and height, its depth, border width, class, color-map identifier (if any), map state, backing-store hint, and location of the corners.

-tree Like option **-children**, but displays all children recursively, even unto the last generation.

-wm Display the window manager's hints. These can include whether the application accepts input; what number and name of the the window's icon; where the window's icon should go; and what the window's initial state should be.

Environment

xwininfo reads the environmental variable **DISPLAY** to get the default host and display number.

See Also

xprop, X utilities

Notes

The **-geometry** string displayed must make assumptions about the window's border width and the behavior of the application and the window manager. As a result, the location given is not always correct.

xwininfo was written by Mark Lillibridge of MIT Project Athena.

xwud — X Client

Un-dump a window image

xwud [-bg *color***] [-display** *display***] [-fg** *color***] [-geometry** *geometry***] [-help] [-in** *file***] [-new]**
[-noclick] [-plane *number***] [-raw] [-rv] [-std** *maptype***] [-vis** *vis-type-or-id***]**

The X client **xwud** un-dumps a window image that had been stored by the X client **xwd**.

xwud recognizes the following command-line options:

-bg *color* Specify the color to display for the '0' bits in the image if a bit-mapped image (or a single plane of an image) is being displayed.

-display *display*
Display the un-dumped image on *display*.

-fg *color* Specify the color to display for the '1' bits in the image if a bit-mapped image (or a single plane of an image) is being displayed.

-geometry *geometry*
Specify the size and position of the window. In most instances, you will want to specify only its position, and let the size default to the actual size of the image.

-help Print a short description of these options.

-in *file* Read the image from *file*. If its command line names no *file*, **xwud** reads its input from the standard input.

0w Create a new color map for displaying the image. If its image characteristics match those of the display, this option can move the image onto the screen more quickly, but at the cost of using a new color map (which, on most displays, scrambles the colors on the other windows).

-noclick By default, clicking any mouse button while the mouse cursor is within the window terminates the application, and so erases the un-dumped image from the screen. This option turns off that behavior. To terminate the program when it is in no-click mode, type **q**, **Q**, or **<ctrl-C>**.

-plane *number*
Display a single bit plane of the image. Bit planes are numbered, with zero being the least significant bit; for example, a 256-color image consists of eight bit planes, numbered 0 through 7. You can use this option to figure out which plane to pass to **xpr** for printing.

-raw Display the image with whatever color values happen to exist on the screen. This option is useful when un-dumping an image onto the screen from which it was originally dumped, while the original windows are still on the screen. This helps move the image onto the screen more quickly.

-rv Invert the meaning of each bit: change every 0 into a 1, and vice versa. This option swaps the foreground and background when displaying a bit-mapped image or a single plane of an image.

-std *maptype*
Display the image using *maptype* as the standard color map. The property name is obtained by converting the type to upper case, and appending to it the prefix **RGB_** and the suffix **_MAP**. Typical types are **best**, **default**, and **gray**. See the Lexicon's entry for **xstdcmap** for one way of creating a standard color maps.

-vis *vis-type-or-id*
Specify visual or visual class. The default is to pick the "best" one. A particular class can be specified, e.g., **StaticGray**, **GrayScale**, **StaticColor**, **PseudoColor**, **DirectColor**, or **TrueColor**. You can also specify **Match**, which tells **xwud** to use the same class as the source image. Alternatively, an exact visual identifier (specific to the server) can be specified, either as a hexadecimal number (prefixed by "0x") or as a decimal number. Finally, you can specify **default**, which tells **xwud** to use the same class as the color map of the root window. Case is not significant in any of these strings.

Environment

xwud reads the environmental variable **DISPLAY** to find the display onto which it is to un-dump the image.

Files

WDFile.h — Define the format of the X Windows dump file

See Also

xpr, xstdcmap, X clients, xwd

Notes

xwud was written by Bob Scheifler of the MIT X Consortium.

Index

to _

.xmodmaprc 143

/dev/kmem 137
/usr/X11/bin 18
/usr/X11/doc 18
/usr/X11/include/X11 18
/usr/X11/include/X11/bitmaps 19
/usr/X11/include/X11/extensions 19
/usr/X11/include/X11/sys 19
/usr/X11/include/X11/Xaw 19
/usr/X11/include/X11/Xmu 19
/usr/X11/lib 19
/usr/X11/lib/app-defaults 19
/usr/X11/lib/config 19
/usr/X11/lib/fonts 19
/usr/X11/lib/nls 19
/usr/X11/lib/twm 19
/usr/X11/lib/x11perfcomp 19
/usr/X11/lib/xinit 19
/usr/X11/objs 19

A

Adams, Chuck 120
Angebranndt, Susan 98, 105
app-defaults 21
application 31
APPLICATIONS 17
appres 34, 51
AT&T 20
Athena widget set 18, 21, 64
atobm 51
atom 36, 138
autoraise 11

B

background 7
  modify 158
BDF 33
bdftopcf 33, 52
bit map 22, 31
bit-mapped image 51, 53
bitmap 31, 51, 53
bitmap distribution format 33
bitmapFilePath 77
bmtoa 51, 60
Bock, Angela 148
BorderColor 27
BorderWidth 26
bounding box 148
Bovey, John 181
Bradley, John 115
Brunhoff, Todd 70
button
  screen 7

C

calculator 17, 38
Capo, Jose 148
cat 16, 180
Chapman, Ross 138
chase 15
Chee, Dana 136
Chong, Danny 138
clicking 6
client 31, 37, 98
  definition 20
Cline, Ernest 177
clipboard 115
Color 27
color 22, 32
  palette 23
  set mode in Xconfig 23
COLUMNS 74
compiling X applications 43
composite widgets 21
Converse, Donna 115, 159
cul-de-sac 40
cursor
  image of cursors 128
  names 128
cursorfont.h 27, 128
Cursors 26
cut 36, 121, 125
cut buffer 121
cutting 170

D

de-iconify a window 8
defaults 21
Della Fera, Tony 119, 137, 182
DontMoveOff 25
dpi 147
dragging 6
Drewry, Raymond 98
Durbin, E. Mike 148

E

edit bit-mapped image 53
editres 60
emulate3buttons 125
event 35, 126
Everson, Phill 178

F

focus 11
font 22, 32
  bitmap distribution format 33
  choose 41
  cursor 22
  portable compiled format 33
  select 129
font path 22
FontPath 33, 97
fonts 97
fonts.alias 22
fonts.dir 22, 33, 71
foreground 7
Frame 27
Friedman, Mike 45
Fulton, Scott 70, 92, 95, 107, 110, 123, 126, 129, 136, 138-140, 145, 177

G

games . . . 37
geometry . . . 21, 25
Gettys, Jim . . . 137, 155, 177
Greening, Daniel R. . . . 178
Gretzinger, Michael R. . . . 148
gridTolerance . . . 56
GUI . . . 18, 20

H

Hartmann, K. Shane . . . 137
Hess, Richard . . . 71
Hewlett-Packard . . . 17, 38
HP-10C . . . 110

I

ico . . . 37, 66
icon . . . 6
  name . . . 180
iconify window . . . 8
image, bit-mapped . . . 51
  edit . . . 53
imake . . . 19, 33, 43, 67
Imakefile . . . 43
importing X applications . . . 43
Internet . . . 14
ISO Latin 1 . . . 130
ISO-8859-1 . . . 130

J

Jones, Ollie . . . 138

K

Kamens, Jonathan . . . 177
Karlton, Phil . . . 105, 140, 155
Karlton, Philip . . . 98
Kent, Chris . . . 105
keyboard
  mapping . . . 37
kmem . . . 137
Krikorian, David . . . 157

L

LaStrange, Tom . . . 92
Lemke, Dave . . . 71
Lexicon
  introduction . . . 49
libX11.a . . . 46
libXbsd.a . . . 46
Lillibridge, Mark . . . 129, 140, 152, 159, 183
listres . . . 34, 70
LN03 . . . 147
logo . . . 137

M

mailempty . . . 40
mailfull . . . 40
make . . . 33, 43
makedepend . . . 34
Makefile . . . 43
Malone, Stuart A. . . . 137
Mankins, Dave . . . 119
Marguilies, Adam . . . 178
Matic, Davor . . . 52, 60, 142
maze . . . 37, 70
McBeath, Jim . . . 45
McCormack, Joel . . . 105-106, 177
McNamara, Bob . . . 177
Medwin, Larry . . . 133
menu . . . 9
  APPLICATIONS . . . 17, 29
  Properties . . . 11
  TWM Operations . . . 13
  WINDOW OPS . . . 11
MenuBackground . . . 30
MenuForeground . . . 30
MicroEMACS . . . 15, 36
Microsoft Windows . . . 5
Mink, Doug . . . 177
MIT . . . 1, 19
MIT-MAGIC-COOKIE-1 . . . 96
mkdirhier . . . 34, 71
mkfontdir . . . 22, 33, 71, 97
Monochrome . . . 30
Moraes, Mark . . . 106
Motif . . . 20-21
mouse
  click . . . 6
  drag . . . 6
mouse buttons . . . 6
mouse cursor
  definition . . . 6
Moy, Edward . . . 74, 119, 177
MS-DOS . . . 20

N

name
  window . . . 180
Newman, Ron . . . 177
Newman, Todd . . . 98
nm . . . 46
nohup . . . 15
NoTitle . . . 25

O

object-oriented program design . . . 20
oclock . . . 38, 72
OOP . . . 20
OPEN LOOK . . . 20-21
Open Software Foundation . . . 20
OReilly & Associates . . . 2

P

Packard, Keith . . . 73, 92, 98, 117
palette . . . 23
paste . . . 36, 121, 125
pasting . . . 170
Payne, Dave . . . 92
PCF . . . 33
Peterson, Chris . . . 66, 117, 125, 137
Pitschke, Steve . . . 92, 177
pixel . . . 6
PixMap . . . 40
plaid . . . 24
pointer
  definition . . . 6
portable compiled format . . . 33
PostScript . . . 148

PRIMARY 37, 121, 170
primary selection 121, 141
Properties 11
property
  definition 34
  PRIMARY 37, 121, 170
Pulley, Harry 181
puzzle 17, 37, 73

R

Reid, Loretta Guarino 177
resize 36, 74
resource 21, 34, 39
  bitmapFilePath 77
  definition 39
  font 41
  list 70
  print 51
resource name 180
RESOURCE_MANAGER 35
rgb.txt 22, 27, 32
Riddle, Paul 45
root window 21, 23, 33
root window, modify 158
Rosenstein, Mark 115
Rosenthal, David 145
ROWS 74
Rupp, Larry 148

S

Scheifler, Bob 98, 136, 148, 155, 157, 177, 184
Scheifler, Jim 137
Schmidt, Dan 134
screen
  background 33
screen button 7
SCREEN_RESOURCES 35
scrollbar 180
security 96, 172
Serisky, Dave 177
server 20, 95
sessreg 36
shell 14
Shortley, Martyn 178
showrgb 32, 74
shutting down X 18
SIGHUP 97
SIGTERM 97
SIGUSR1 97
SIGWIN 74
slider 9
Solomon, Marvin 148
spoof 172
start.twmrc 77
startx 23, 36, 74, 135
Sternlicht, Dave 92, 142
Sun Microsystems 20
Swick, Ralph R. 110, 117, 123, 132, 177
system.twmrc 19, 25

T

Tallot, Didier 178
Tektronix 4014 14
Texas Instruments 17, 38
TI-30 110
title 180
title bar 7
toolkit 21
troff 148
twm 8, 36, 75, 97
TWM Operations 13
twmrc 19, 77

U

unfocus 12
utility 20, 31, 98
utmp 36

V

Vandevoorte, Mark 74, 177
VGA 20
  palette 23
vi 15, 36
viewres 92
VT-100 . . . . . 14, 38, 178

W

Weiss, Martin 71
Weissman, Terry 177
widget 39
  Athena set 64
  bitmap 59
  composite 21
  editor 60
widget class 21, 39
widget set 21
window
  background 7
  de-iconify 8
  foreground 7
  iconify 8
  names 180
  title 180
window manager
  definition 21
WINDOW OPS 11
workstation 20
write 15, 179
Wyatt, William F. 182

X

X 95
  architecture 20
  history 19
  recompile 43
  releases 19
  revisions 19
  shut down 18
  utilities 31
  window manager 21
X client
  definition 20
X clients 98
X Logical Font Description 129
X server 95, 20
X terminal 20
X utilities 98
x11perf 100
x11perfcomp 105

xauth 35, 106
Xaw 64
xbiff 17, 24, 38, 40, 107
XCalc 39
xcalc 17, 21, 38, 110
xclipboard 36, 115
xclock 16-17, 24-25, 30, 38, 117
xcmsdb 32, 119
xcmstest 120
Xconfig 22, 33, 97
xcutsel 37, 121
XDM-AUTHORIZATION-1 96
xdpyinfo 35, 123
xedit 38, 123
xev 35, 126
xeyes 18, 25, 37, 126
  example of window 75
xfd 22, 27, 33, 127
xfontsel 17, 21-22, 33, 129
xgas 30, 37, 132
xgc 38, 133
xgrabsc 73
xinit 23, 36, 134
xinitrc 19, 23, 74, 134
xkill 37, 136
XLFD 129
Xlib 21
xload 18, 38, 136
xlogo 18, 38, 137
xlogo32 31
xlsatoms 36, 138
xlsclients 36, 139
xlsfonts 33, 140
xmag 18, 38, 140
xmkmf 19, 34, 43, 142
xmodmap 37, 142
xpr 39, 145, 181
xprop 34, 149
xrdb 21, 28, 35, 152
xrefresh 33, 155
xset 33, 95, 97, 156
xsetroot 23-24, 32-33, 158
xstdcmap 32, 159
Xt 21
xterm 39, 97, 160
xtetris 18, 37, 177
xvt 6, 14, 18, 38, 178
xwd 39, 148, 181
xwininfo 36, 182
xwud 39, 181, 183