X Window System User's Guide

OPEN LOOK Edition

Final Draft

Ian Darwin, Valerie Quercia and Tim O'Reilly

Disclaimer

This book was written under a contract to O'Reilly & Associates. Due to the announcement of the Common Desktop Environment (CDE) by Sun, HP, IBM, Novell, and others, and the planned phaseout OPEN LOOK by Sun, O'Reilly decided not to publish the book. Accordingly, it is being released as an unpublished work. It is not being published by O'Reilly & Associates. This means that this version has *not* been subjected to O'Reilly & Associates' usual detailed editing and quality-assurance process. Accordingly, neither O'Reilly & Associates, nor the author, nor Darwin Open Systems assumes any liability for the accuracy of this book, nor for errors & omissions.

Since the book is being distributed on CD-ROM, the manual pages referred to throughout as being "in Section Three of this Guide" are in fact stored in the *man* subdirectory of the CD-ROM instead.

In addition, work on this book was suspended for parts of 1993 and 1994, during a time when it appeared that the work would never see the light of day. Some parts have been updated for 1995, but other sections of the material will not refer to the very latest versions of the software. Finally, since there are so many versions of OpenWindows in circulation, we can't say everything about all of them, so we try to take a middle ground between the very latest and the older ones.We apologize for any discrepancies, but wanted to bring the book to you in the most timely and cost-effective way possible; this has meant sacrificing the task of updating all the screen-shots and descriptions of the menu items in order to bring you a low-cost product during a time when the OPEN LOOK Graphical User Interface is still widely used in the Sun and Linux user communities.

Ian F. Darwin
March, 1995

For my best friend and wife Betty, whose time at my side has vastly enriched my life , and who fed the children and washed the dishes while I was writing this book.

For Benjamin, Andy and Margartet who brought snacks and smiles.

Trademarks

Apple, Apple LaserWriter, Apple LaserWriter Plus, MacIntosh (sp?) of Apple Computer...
Bembo and Gill Sans are trademarks of the Monotype Corporation Plc.
Helvetica is a trademark of Linotype AG and/or its subsidiaries.
ITC Avant Garde Gothic, ITC Bookman, ITC Zapf Chancery, and
ITC and Zapf Dingbats are registered trademarks of International Typeface Corporation.
Lucida is a registered trademark of Bigelow & Holmes.
New Century Schoolbook, Palatino , and Times are trademarks of Linotype AG and/or its subsidiaries.
OPEN LOOK was a trademeark of AT&T and is now a trademark of Novell Inc.
OSF/Motif is a trademark of the Open Software Foundation
PostIt (3M?)
PostScript is a trademark of Adobe
Rockwell is a trademark of The Monotype Corporation Plc.
Sun Workstation, OpenWindows, Sun-3, Sun-4, NeWS, X/NeWS, NeWSprint, AnswerBook, SunPC are trademarks of Sun Microsystems Inc
SPARC is a trademark of SPARC International
Tek and Tektronix are trademarks Tektronic Incorporated.
UNIX has been a trademark of Bell Laboratories, AT&T, Western Electric, Unix System Laboratories Inc., Novell Inc., and others, and is now a trademark of X/Copen Co. Ltd..
X Window System is a trademark of either MIT or The X Consortium

Any other capitalizaed tradename should be presumed to be a trademark of the owner of that product.

Preface

The OPEN LOOK GUI is a popular user interface style (also called a "graphical user interface," or GUI) for programs running on window systems like The X Window System. X11 itself is a network-based graphics windowing system developed at MIT and widely adopted as an industry standard. But X11 only provides the foundation and skeleton of a window system, just as concrete and wood provide the foundation and framework for a house. The material that you put on the outside will significantly affect both how the final edifice looks and how comfortable it will be to live and work in, be that edifice a house or a window system. One exterior, or Graphical User Interface, that became popular in the late 1980's and early 1990's is OPEN LOOK. The OPEN LOOK GUI was designed by Sun and AT&T with help from Xerox, and is based in part—like other X11 GUIs such as OSF/Motif, and other window systems such as the Apple Macintosh—on pioneering work done in the late 1960's and 1970's at Xerox Palo Alto Research Center (PARC) and other research sites. OPEN LOOK has also been influenced by a family resemblance to SunView, Sun's earlier, very successful, and long-popular workstation window system.

If you are new to X, you may be wondering why you should consider the OPEN LOOK GUI. First of all, OPEN LOOK has many features that make it more intuitive and easier to use than other X-based user interfaces. For example, frequently-accessed menus can be "pinned up" for as long as you need them. As well, the graphical elements each have a distinctive shape—a button is always oval, a pull-down menu is always shown by a small triangle, etc.—which makes it easier to spot buttons, pull-down menus, etc., than when using a GUI that uses rectangular shapes for everything. And of course if you have previously used SunView, you will find much that is familiar in the OPEN LOOK GUI.

In addition, the OPEN LOOK GUI specifies the behavior of certain key applications such as the File Manager (other GUIs do not specify this, and thus several different and incompatible "desktop managers" are available at extra cost). Comprehensive implementations such as Sun's OpenWindows include a suite of useful "deskset" applications that make the interface much more than a bare-bones "look and feel." OpenWindows includes (in addition to

the File Manager) a set of useful tools called the DeskSet, which includes a calendar manager for scheduling appointments among people in your workgroup, a multi-media mail tool that can include audio files and graphical images in mail messages, tools for controlling printers, etc. All of these are either not available or cost a significant amount of extra money when using other GUIs.

Because the OPEN LOOK GUI is a specification, there are several implemtnations of it. Some of the complexity of this volume derives from listing the differences amont them The *X Window System User's Guide, OPEN LOOK Edition* covers the most popular implementation of the OPEN LOOK GUI, Sun's OpenWindows Version 3.* Much of the information is also applicable to AT&T's OPEN LOOK 4 software, and indeed much of it is applicable to any version of The X Window System.

We describe the basic concepts of window systems and terminology, the application programs (clients) commonly included with these packages as well as some free-software clients that you may wish to obtain on your own, and how client programs behave under OPEN LOOK. The OPEN LOOK edition is intended for those using X with the OPEN LOOK interface. There are also Motif and "generic (MIT) X11" versions of this book available from O'Reilly & Associates.

Historical Update

The OPEN LOOK Graphical User Interface has been abandoned by its two sponsors, Sun and AT&T, as part of the Common Open Systems Environment process' Common Desktop Environment (CDE). As a result, there is no new commercial development of OPEN LOOK applications. Most commercial software for X is now being deveoped for Motif, since that is the basic GUI specified by CDE. However, the availability of the XView toolkit as free software and its popularity in the Linux field, as well as the large market share of Sun Microsystems workstations, all of which still (March 1995) ship with OPEN LOOK as their default windowing environment, ensures that OPEN LOOK will remain popular for some years to come.

Assumptions

This book assumes that the OpenWindows package or another OPEN LOOK package is already installed on your UNIX system in the standard locations, and that all standard OPEN LOOK clients are available (if not, consult the appropriate vendor's documentation, or refer to Volume Eight, *X Window System Administrator's Guide*). If you are on Linux, then most of the Sun Deskset tools will not be available, though you may be able to run them remotely across the network. This book also assumes that you have basic familiarity with The UNIX Operating System. It is probable that the OPEN LOOK GUI will be "ported" or made available on operating systems other than UNIX; the XView toolkit, for example, has been ported to every major version of UNIX and is commercially available for Digital

* Version 3 is standard on Solaris 1.0 (SunOS 4.1.2) on SPARC, and Solaris 2 (SunOS SVR4) on both SPARC and Intel. Much of the information also applies to OpenWindows Version 2, though the DeskSet clients and the operation of drag-and-drop are different. We do not mention all the differences, but do provide some coverage of OpenWindowsV2 because it's the last version that Sun is making available for the Sun-3 product family line.

Equipment's VMS operating system. And Quantum Software's QNX operating system uses the OPEN LOOK GUI as the interface to QNX Windows. If you are using the OPEN LOOK GUI on an operating system other than UNIX you will still find this book useful—UNIX dependencies are not that widespread—but you may occasionally need to translate a command example into its equivalent on your system. This book also assumes that you are using a three-button mouse, and that the configuration files provided by your vendor for *olwm* and the clients have not been extensively modified locally. (If this is not the case, the book provides information that will allow you to understand how these programs can be configured on your system.)

In a few places we refer to "contributed" software. Most of the OPEN LOOK-specific free software is included with the CD-ROM accompanying this book. If this is not available on your system, you can either ignore the information—none of these programs is critical, though we have included some that are useful—or you can obtain them in source form by *ftp* or *uucp* from UUNET or elsewhere, then use *imake* to compile them. See *Xshowfonts and Other Free Programs* below for details.

This book is written for both first-time and experienced users of the OPEN LOOK GUI on the X Window System. First-time users should read the book in order, starting with Chapter 1, *An Introduction to OPEN LOOK and the X Window System*.

Experienced users of other X11 variants can use this book either to learn the specifics of OPEN LOOK—start with Chapter 2, *Working in the OPEN LOOK Environment*—or as a reference for the client programs detailed here. Since there is great flexibility with X, even frequent users need to check the syntax and availability of options. Reference pages for common client programs list command-line options, customization database (resource database) variables, and other details.

Organization

The book contains these parts:

Part One: Using X and OPEN LOOK

The *Preface* describes the book's assumptions, audience, organization, and conventions.

Chapter 1, *An Introduction to OPEN LOOK and the X Window System*, describes the basic terminology associated with OPEN LOOK and the X Window System: server, client, window, etc. The most important client programs are described.

Chapter 2, *Working in the OPEN LOOK Environment*, shows how to start the programs necessary to begin using X: the server, the first terminal window, and the window manager. It teaches the skills necessary to begin working productively. This chapter also shows you how to add additional windows; exit from a window; use the tools of the display; do some basic window manager operations using the olwm window manager, and set up the display.

Chapter 3, *Opening Additional Windows*, describes how to start additional client windows. You are introduced to two methods of customizing X client programs: command-line

options and resource variables. In addition, Chapter 3 shows you how to identify OPEN LOOK clients, X Toolkit clients, and Motif clients.

At this point you will want to start *using* the system. If you prefer a visual, icon-oriented method of using computers, you should learn to use the File Manager. If you are familiar with the UNIX command-line interface, you should learn to use the *cmdtool* terminal emulator.

Chapter 4, *Using the OPEN LOOK File Manager*, discusses the OPEN LOOK File Manager, a comprehensive user-interface program. It shows the small differences between the Sun and AT&T versions.

Chapter 5, *The cmdtool/shelltool Terminal Emulator*, tells you how to use Sun's terminal emulator. Certain aspects of its operation apply to other OPEN LOOK clients as well, particularly *textedit*(1) and to all OPEN LOOK clients that have "text fields." For this reason, *textedit*(1) is described in this chapter. The other commonly-used terminal emulator, *xterm*, is described in Appendix A, *The xterm/olterm Terminal Emulator*.

Chapter 6, *Using the OPEN LOOK Window Manager*, describes additional window manager operations, such as resizing windows and changing the order of windows in the stack on your screen. It also describes the minor differences between the Sun and AT&T versions.

Chapter 7, *The OpenWindows DeskSet Clients*, describes the DeskSet clients that Sun's OpenWindows package includes: Multimedia Mail Tool and Audio Tool that let you record, play back, and send and receive verbal or other sound data; the Calendar Manager for scheduling individual or group appointments, *print tool, tape tool, perfmeter, clock, calculator, iconedit, binder*, and two online documentation facilities, Answerbook and the Help Viewer.

The following programs, though strictly speaking part of the DeskSet, are described elsewhere: *File Manager* (described in Chapter 4, *Using the OPEN LOOK File Manager*), *cmdtool/shelltool* and *textedit* (described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*), and *snapshot* (described in Chapter 9, *Graphics Clients)*.

Chapter 8, *Other Standard Clients*, gives an overview of other clients available with standard distributions of The X Window System, including window and display information clients, the *xkill* program, and several "desk accessories." In describing various MIT standard clients, this chapter highlights some features of the Athena widget set.

Chapter 9, *Graphics Clients*, explains how to use the major graphics clients included with The X Window System and OPEN LOOK and OpenWindows. The OPEN LOOK programs include the *snapshot* and *olprintscreen* screen capture programs, and the *olpixmap* and *iconedit* icon editors. The standard clients include *xwd, xpr*, and *bitmap* . The chapter ends with sections on PostScript previewing and font editing.

Part Two: Customizing X

Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, describes the somewhat complicated font naming conventions and ways to simplify font specification, including wildcarding and aliasing, for both regular and scaleable fonts. It describes how

to use the *xlsfonts*, *xfd*, and *xfontsel* clients and the OpenWindows *text* demo to list, display, and select available display fonts.

Chapter 11, *Command-line Options*, discusses some of the standard command line options accepted by most X11 clients, be they MIT clients or OpenWindows clients.

Chapter 12, *Setting Resources*, shows you how to create an *.Xdefaults* file to set default characteristics for client applications. Also discusses the tool (called *Properties...* on OpenWindows and "Workspace Manager" on AT&T-OL) that maintains your *.Xdefaults* file for you. This chapter also describes how to use *xrdb*, which saves you from having to maintain multiple *.Xdefaults* files if you run clients on multiple machines.

Chapter 13, *Customizing olwm*, describes the process of customizing *olwm* by showing the default configuration files shipped by Sun and AT&T, and examines the purpose and syntax of entries. It explains techniques for revising the files to modify existing menus and create new ones. Chapter 13 also reviews the X resources that can be used to control *olwm*.

Chapter 14, *Customization Clients*, describes how to set display and keyboard preferences using *xset* and how to set root window preferences using *xsetroot*. Chapter 14 also demonstrates how to use *xmodmap* to redefine the logical key names and pointer commands recognized by X. In addition, we describe some general customizing utilities, such as binder, which updates the file manager's list of icons and programs.

Part Three: Client Reference Pages

Presents extended reference pages for three versions of the server (MIT, AT&T and Sun) as well as all clients mentioned in this manual. These are in the standard UNIX "manual page" format and should be familiar to users of The UNIX Operating System or UNIX-like systems; for others, the format is described briefly at the beginning of this Part.

Historical Note

The Client Reference Pages are not included in the body of this book, but most of them are included in the *man* subdirectory of the CD-ROM accompanying this book.

Part Four: Appendices

Appendix A, *The xterm/olterm Terminal Emulator*, describes how to use the *xterm* terminal emulator. Certain aspects of *xterm* operation described in this chapter, such as scrolling and "copy and paste," are common to other X Toolkit applications as well.

Appendix B, *OpenWindows and X11 Fonts*, discusses the fonts included with OpenWindows and standard X11 releases, and shows how to add your own fonts.

Appendix C, *Standard Bitmaps - X11, OPEN LOOK and OpenWindows*, describes the standard bitmap images that you can use for showing as icons, in your root window, etc.

Appendix D, *Standard Cursors - X11 and OPEN LOOK*, shows the standard cursors, and describes what some of them mean.

Appendix E, *Control Sequences for xterm and cmdtool*, describes the "escape sequences" used to control the *cmdtool* and *xterm* terminal emulators.

Appendix F, *X Toolkit Translation Table Syntax*, deescribes the translation tables used by Xt libraries—OLIT and Motif.

Appendix G, *Introduction to Xt Widget Resources*, gives an introduction to the ways that Xt toolkits such as OLIT fit "Widgets" together.

Appendix H, *OPEN LOOK Intrinsics Toolkit Widget Resources*, is not included in this book for reasons of copyright. But most of the programs we discuss are based on XView, so the material in Appendix J will be more relevant anyway. If you are using OLIT, you have the manual pages with your toolkit that describe the X Resources they use..

Appendix I, *Athena Widget Resources*, describes the Resources used by the standard MIT clients.

Appendix J, *OPEN LOOK XView Toolkit Resources*, describes the resources used by the OPEN LOOK XView toolkit.

Appendix K, *OPEN LOOK Mouseless Operations*, summarizes the editing keys that can be used to perform window operations from the keyboard.

Appendix L, *Running SunView Applications on OpenWindows*, discusses how SunView applications differ from X applications under OpenWindows

Appendix M, *OPEN LOOK and Motif*, describes how to use Motif applications in relative harmony with OPEN LOOK applications. Discusses both Motif applications under the OPEN LOOK Window Manager and OPEN LOOK applications under the Motif Window Manager, *mwm.*

Appendix N, *OPEN LOOK Software Availability*, describes the availability of software that implements and uses the OPEN LOOK GUI.

The *Glossary* provides a list of the important technical terms and their meanings.

The *Documentation Roadmap and Bibliography* will describe the vendor documentation and some additional reading.

Index

On the Complexity of X

One complaint you'll hear over and over about the X Window System is that it's too complicated. One reason for this attitude is that X is widely available, and as a result many programmers have turned their hand to extending it. As a result of that, there are many different ways to do even the simplest thing. We make no apology for here documenting several ways to do most everything, as it is as yet too early in the evolution of X to tell which will survive and become widely used. Yes, X appears complex because it has many variations. But you seldom need to know more than one way of doing something, so we offer this advice to the X-struck:

Try out a few ways of doing something. Remember the one that works best for you; forget all the rest.

xshowfonts.c and Other Free Programs

There is a lot of free or contributed software to accompany The X Window System. Indeed, Sun Microsystems has contributed the source for their XView toolkit and several accompanying clients. The two most important sources of this software in North America are UUNET—a non-profit networking site established by the USENIX Association—and *export.lcs.mit.edu*, an anonymous-*ftp* site set up by The X Consortium.

The source to *xshowfonts.c*, which is printed in Appendix C, *Standard Bitmaps – X11, OPEN LOOK and OpenWindows*, is included in the CD-ROM that accompanies this book. If you have received only a printed copy of the book, this program is also available free from Uunet (although there is a small connect-time charge if you use their dial-up service). If you have access to UUNET, you can retrieve the source code using *uucp* or *ftp*. For *uucp*, find a machine with direct access to UUNET, and type the following command:

```
uucp uunet!~uucp/published/nutshell/Xuser/xshowfonts.c.Z \
        yourhost!~/yourname
```

The exclamation marks (!) wil need to be escaped with a backslah if you use the C-shell (csh) instead of the Korn or Bourne shell. The file should appear some time later (up to a day or more) in the directory */usr/spool/uucppublic/**yourname***.

You don't need to subscribe to Uunet to be able to use their archives via *uucp*, within the U.S.A. By calling 1-900-468-7727 and using the login "uucp" with no password, anyone may *uucp* any of UUNET's on-line source collection. Start by copying *uunet!~/ls-lR.Z*, which is a compressed index of every file in the archives. Peruse this file to get an idea of the programs available. The file *uunet!~/published/nutshell/ls-lR.Z* contains a listing of the files in the *published/nutshell* subdirectory (example programs for our books). As of this writing, the cost is 40 cents per minute. The charges will appear on your next telephone bill.

It is more common today to access UUNET vit the Internet. To use *ftp*, you will need to find a machine with direct access to the Internet. The following example is a sample session, with commands in boldface.

```
% ftp ftp.uu.net
Connected to ftp.uu.net.
220 uunet FTP server (Version 5.99 Wed May 23 14:40:19 EDT 1990)
ready.
Name (ftp.uu.net:ambar): anonymous
331 Guest login ok, send ident as password.
Password: vol3ol@ora.com     (use your user name and host here)
30 Guest login ok, access restrictions apply.
ftp> cd published/nutshell/Xuser
250 CWD command successful.
ftp> binary  (you must use binary transfer for compressed files)
200 Type set to I.
ftp> get xshowfonts.c.Z
200 PORT command successful.
```

```
150 Opening ASCII mode data connection for xshowfonts.c.Z (5587
bytes).
226 Transfer complete.
ftp> quit
221 Goodbye.
%
```

The file is a compressed C program. To uncompress the program, type:

```
% uncompress xshowfonts.c
```

The *compress*(and *uncompress*)programs are now shipped with most vendors' versions of UNIX and are included in System V Release 4. If you do not have them, you can get the source code (in uncompressed form, of course) from Uunet.

Finally, if you are on the Internet, the machine *export.lcs.mit.edu* is the X Consortium's public *ftp* machine; it includes the official (latest release) distribution, as well as a terrific amount of contributed software. There are other X archive sites around the Internet; consult the Frequently Asked Questions article in the mailing list *xpert@expo.lcs.mit.edu* or the USENET group *comp.windows.x* for an up-to-date list of archive sites.

These programs normally come in compressed form (as the example above) or in "compressed tar" form. The general assumption is that a C programmer will unpack, compile and install them. But it's not that hard to do this if you are not a programmer. A simple recipe is include in the book *X Window System Administrator's Guide* described below.

X Window System Administrator's Guide, with CD-ROM

X11 sources are also available bundled with a copy of Volume Eight, *X Window System Administrator's Guide* (O'Reilly & Associates, 1992). CD-ROM drives are now available for just about every workstation. Many UNIX vendors now distribute their operating systems only on CD, requiring you to buy a player. SunOS, DEC Ultrix, and SGI IRIX are currently shipping on CD-ROM, as are (optionally) MicroSoft Windows and hundreds of commercial applications, with more vendors expected to follow.

In fact, CD-ROM drives are now cheap enough that it's worth getting a CD-ROM drive and this disk even if you already have the full source of X11, because the magnetic disk space you free up can be put to other, more productive purposes.

The CD includes:

- "Rock Ridge" CDROM drivers from Young Minds, so you can install the CD as a UNIX filesystem on several UNIX platforms (current releases of SunOS already include Rock Ridge format CD's).
- Complete "core" source for MIT X11 Release 4 and 5. This includes the new R5 features, such as the font server and XCMS, the color management system.
- Complete "contrib" source for MIT X11 Release 5. This includes the complete source for the XView OPEN LOOK toolkit and several sample clients, as well as some programs not in the MIT distribution, such as *xtici*, the Tektronics Color Editor.
- Complete examples and source code from all the books in the X Window System Series.

- Programs and files that are discussed in Volume Eight. These were previously available only to administrators with Internet access.
- Pre-compiled X11 Release 5 binaries for Sun3 and Sun4 platforms running SunOS 4.1.1.

The *X Window System Adminstrator's Guide* includes directions for building and installing the MIT sources, so even if you don't currently have X up and running, you can do so with this book/CD package. Contact O'Reilly & Associates or your local bookseller.

Other Books in this series

The O'Reilly X Guides provide a comprehensive set of documents for The X Window System. Other books in the series which may be of interest to end-users include:

X Window System Administror's Guide (Volume 8). As mentioned above, this volume discusses many aspects of setting up and running the X Window System, particularly in a multi-vendor environment.

X Window System in a Nutshell. (Nutshell series) Contains reference pages for many of the standard clients, along with programmer documentation on the system.

Most of the remaining books in the series are of interest to programmers developing and maintaining X Window System applications:

Volume Zero, *X Protocol Reference Manual*, discusses the network protocol used between the X server and its clients. Of interest mainly to advanced programmers.

Volumes and Two, *XLib Programmer's Manual* and *Xlib Reference Manual* discuss the use of XLib, the lower level of access.

Volumes Four and Five, X Toolkit Intrinsics Programmer's Manual. and *X Toolkit Intrinsics Reference Manual*, discuss the details of the "Xt" toolkit library.

Volume Seven, *XView Programming Manual*, discusses Sun's XView toolkit for building OPEN LOOK applications.

Volume Six, *Motif Programming Manual*, discusses in full detail the Motif toolkit.

PHIGS Programmer's Manual. (X series, no volume number) discusses in considerable detail the use of PHIGS graphics and PEX, the Phigs Extension to X, a graphics layer that can be used on top of X.

Power Programming with RPC. (Nutshell Handbook series) discusses developing networked applications using Sun's Remote Procedure Call library, including significant discussion of developing RPC applications that work within the constraints of X toolkits such as Sun's OPEN LOOK toolkit XView and Xt (including OLIT and Motif).

Bulk Sales Information

This series of guides is routinely resold by many Workstation and X Terminal manufacturers as their official X Window System documentation. For information on volume

discounts for bulk purchases, call O'Reilly & Associates, Inc., at 800-338-6887 (other numbers below), or send e-mail to *linda@ora.com* (*uunet!ora!linda*).

We will do selected customization for you. For companies requiring extensive customization of the guide, source-licensing terms are also available.

Request for Comments

Please write to tell us about any flaws you find in this manual or how you think it could be improved. This will help us to provide you with the best documentation possible.

Our U.S. mail address, phone numbers, and e-mail addresses are as follows:

O'Reilly and Associates, Inc.
632 Petaluma Avenue
Sebastopol CA 95472
U.S.A.

In USA 1-800-338-6887, in California 1-800-533-6887, international +1 707-829-0515

UUCP: *uunet!ora!ian*
Internet: *ian@ora.com*

Font and Character Conventions

These typographic conventions are used in this book:

Italics are used for:

- new terms where they are defined.
- file and directory names, and command and client names when they appear in the body of a paragraph.

`Courier` is used within the body of the text to show:

- command lines or options that should be typed verbatim on the screen.

and is used within examples to show:

- computer-generated output.
- the contents of files.

`Courier bold` is used within examples to show command lines and options that should be typed verbatim on the screen.

`Courier italics` are used within examples or explanations of command syntax to show a parameter to a command that requires context-dependent substitution (such as a variable). For example, ***`filename`*** means to use some appropriate filename; ***`option(s)`*** means to use some appropriate option(s) to the command.

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

[ ] surround an optional field in a command line or file entry.

name(1) is a reference to a command called *name* in Section 1 of the *UNIX Reference Manual* (this book may have a different name depending on the version of UNIX that you use). The OPEN LOOK-specific commands are reprinted in Part Three of this Guide.

$ is the standard prompt from the Bourne shell, *sh*(1).

% is the standard prompt from the C shell, *csh*(1).

Since there are some variations in the commands available with the various implementations of OPEN LOOK, we use the following tags:

AT&T-OL refers to a feature of the AT&T OPEN LOOK client package

OpenWindows refers to a feature of Sun's OpenWindows client/server package.

Acknowledgments

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

The OPEN LOOK version of this *User's Guide* was written by Ian Darwin, but the overall design, as well as some parts (even a few whole chapters) of the text, were either adapted or borrowed intact from the "Generic X11" and Motif editions. The OPEN LOOK version was developed using Sun's OpenWindows 2., 3.0 and 3.1on Sun-3 Workstations and SPARCstations and on an AT&T 6386 WGS with AT&T's XWIN and OPEN LOOK product. Earlier versions of this guide were developed on a Sony NEWS workstation running Sony's X implementation, a Visual 640 X Display Station, and an NCD16 Network Display Station.

Mr. Darwin wishes to express his gratitudel to Sun Microsystems for ongoing provision of software, documentation, and technical support, and to AT&T and Quantum Systems for providing copies of their OPEN LOOK software and QNX-Windows systems respectively. We appreciate the support of these manufacturers in helping us develop complete and accurate X Window System documentation. Thanks are particularly due to Brad Keates of Sun Canada, Roger Nolan and Andrew Barker of Sun Microsystems, and Joanne Newbauer of AT&T. We'd also like to thank Sun for distributing the XView toolkit with an accompanying Copyright that grants permission to reprint the reference manual pages elsewhere in the CD-ROM, and to excerpt parts of the material (in particular, Appendix J, *OPEN LOOK XView Toolkit Resources*). Most of the remaining reference manual pages, those for standard MIT X clients, have been adapted from reference pages copyright © 1990-1994 the Massachusetts Institute of Technology, the X Consortium, and others. Refer to the "Authors" section at the end of each reference page for details. Other copyrights are listed on the relevant reference pages.

Mr. Darwin would like to thank the previous authors for providing the framework of this book. He also thanks the entire staff of O'Reilly and Associates for helping to produce the draft version of this book, with special thanks to Tim O'Reilly for patience above the call: this book took much longer to complete than he and I *combined* ever thought it would. "No man who is in a hurry is quite civilized." —Will Durant. The initial book design was done (initially for the *PHIGS Programming Manual* in this series) by Bill Prindle and Edie Freedman and implemented by Mike Sierra. Mike also provided invaluable assistance in learning the hidden craft of using FrameMaker. Dale Dougherty provide the initial version of the *troff*-to-Frame converter, and offered ongoing support while it was extended. Donna Woonteiler read a draft of the whole book with a careful eye to editorial regularity. Linda Walsh and Tim O'Reilly cooperated in getting clearance to reprint this work in CD-ROM form in 1995. Mr. Darwin thanks these and everyone else at O'Reilly for their unflagging assistance in bettering this book

Valerie Quercia of O'Reilly and Associates and Chuck Musciano of Harris Corporation made constructive comments on the Alpha Draft of the OPEN LOOK version of this work; their help is gratefully acknowledged.

And thanks to the staff of SoftQuad, Inc., for lending their support and for careful proofreading of many parts of the book. Liam Quin (who wrote and maintains the "Frequently Asked Questions" list on the USENET group *comp.windows.open-look*) was especially helpful.

Mr. Darwin also thanks Ken Thompson, Dennis Ritchie *et al* of the AT&T Bell Laboratories Computer Science Research department for inventing, promulgating, and extending the UNIX system; all the folks at MIT Project Athena (where X originated), the X Consortium, and elsewhere who have contributed to the design, implementation and spread of The X Window System. Special thanks to the AT&T and Sun team that originated the OPEN LOOK GUI specification.

For significant help and favors along the way, Mr. Darwin thanks Geoff Collyer, Erik Fair, Brian Kernighan, and Al Lambert. John Gilmore and Laura Creighton first got me interested in SunView, the predecessor in the Sun UNIXverse of The X Window System.

Finally, thanks to my wife Betty and children Benjamin, Andy, and Margaret for sparing me the time to write all this. Behind every parent who writes, there must be (or ought to be) a spouse as supportive as Betty. In addition, her proofreading skills salvaged scores of sick sentences.

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

PART ONE: Using X with OpenWindows

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 1

An Introduction to OPEN LOOK and X Windows

This chapter describes the features of a typical display running OPEN LOOK on the X Window System, and introduces some basic window system concepts. It also provides an overview of the X Window System's client-server architecture and briefly describes some common clients.

The OPEN LOOK Graphical User Interface (GUI) gives display-based programs an interface that is efficient, easy to use, and consistent. OPEN LOOK is supported by Sun, by AT&T, by many system and application vendors, and is the standard graphical interface with the current release of UNIX, System V Release 4.

The X Window System, called X for short, is a network-based graphics window system that was first developed at MIT around 1984. Several versions of X have been developed, the most recent of which is X Version 11 (X11), first released in 1987.There have been several revisions of X11 since then. The latest is Release 6, or "X11R6".

X11 has been adopted as an industry standard windowing system. It's supported by every significant workstation vendor, and its development is now managed by The X Consortium, a group of UNIX and X industry leaders such as Sun, DEC, Hewlett-Packard, IBM, NCD, and many others who have united to direct, contribute to, and fund its continuing development. In addition to the system software development directed by the X Consortium, many programmers develop "contributed" or free software for X, and many software developers are producing commercial application software specifically for use with X, including spreadsheets, graphics and database programs, and publishing applications.

But the original versions of X11 did not provide or specify any graphical user interface. X is a basic window system upon which almost any style of GUI can be built. Even "The X Toolkit" provides only a simplified approach to creating graphical user interface components—guidelines for writing and implementing widgets—rather than offering a set of components with a predefined look and feel.

1

When X first started to become popular, in fact, it was so ugly that one old-timer said: "X [as it was then, with] each program having its own randomly-chosen conventions, looks like a ransom-note user interface system. This was certain to drive people away from X and UNIX to the Macintosh."

The X developers at MIT took a hands-off approach; they did not want to step into the GUI standards vacuum. Their battle cry was (then, and for many years after) "Mechanism, not Policy." That is, X itself has, and will always have, the flexibility to support any number of user interface styles.

To fill the standards gap, Sun and AT&T developed the OPEN LOOK GUI specification with input from many others in the industry. The OPEN LOOK document specifies not only the shape of various buttons and gadgets on the screen, but also the operation of various important clients, such as the File Manager (see Chapter 2, *Working in the OPEN LOOK Environment*, and Chapter 4, *Using the OPEN LOOK File Manager*). They published the specification quite early, so that other vendors could work to it, making OPEN LOOK the leading open standard for graphical user interface.

The design they came up with is clean, simple, and intuitive. My son Ben, then three and a half years old, climbed up on my lap one day, pointed at the scrollbars (graphic objects used to move or "scroll" a large data window around in a smaller on-screen window). Without puzzling over graphic design for a second, he asked me "What are those elevator things for"? And before long he was rearranging the windows on my screen. So even a three year old can learn to use OPEN LOOK: the design is clean, simple, and intuitive.

Shortly after Sun and AT&T announced that they would be working together on the development of OPEN LOOK, the Open Software Foundation announced its own development of a "competing standard" GUI. This led to the eventual release of OSF/Motif, comprising a set of "widgets" (graphical objects) made by blending together widget sets provided by DEC and HP, and a window manager *mwm*. It seemed clear that both OPEN LOOK and Motif would be with us as "competing standards" for most of the 1990's.†

However, in 1993, some of the leading vendors got together and agreed on a new set of standards. The COSE group produced one, the Common Desktop Environment, that was based primarily on Motif. Thus it is likely that OPEN LOOK's importance will diminish over time. However, most COSE vendors do not plan to begin shipping the Common Desktop Environment until late in 1995, and in the meantime many people are using X and OPEN LOOK...

This book is about OPEN LOOK and X Windows. So we'll start this chapter by looking at a typical X display and consider some general features of the system. We'll also briefly compare a standard X application (written with the X Toolkit) to an OPEN LOOK applica-

† The OPEN LOOK specification, and all implementations, include these features not in Motif 1.1, many of which OSF is hoping to fit in to their next release: context-sensitive help, pinnable menus, and drag-and-drop. It is to be hoped that the OSF will provide these features in ways that are compatible with the widely-used "prior art" provided by the OPEN LOOK implementations features. OPEN LOOK also specifies the behavior of important clients such as the File Manager, Properties/Workspace manager, and others; these are described in upcoming chapters of this guide.

tion. Then we'll discuss what distinguishes the X Window System from other window systems. We'll also introduce some of the more important programs included in the standard distribution of X, and the *olwm* window manager shipped with OPEN LOOK. In the next chapter we'll show you how to start your X session and how to work in the OPEN LOOK environment.

1.1 Anatomy of an X Display

X is typically run on a workstation with a large screen. The X Window System also runs on PCs with 80386 CPUs and VGA graphics, on special X terminals, and on many larger systems. X allows you to work with multiple programs simultaneously, each in a separate *window*. The display in Figure 1-1 includes several windows.

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

The windows you will probably use most frequently are *terminal emulators*, windows that function as standard terminals. The terminal emulator included with the MIT release of X is called *xterm*; an OPEN LOOK standardized version of this program is called *olterm* on some systems. The terminal emulator used by default under OpenWindows is called *cmdtool* or *shelltool*. Figure 1-1 depicts two *xterm* windows and one *cmdtool* window. In a terminal emulator window, you can do anything you might do when logged in at the console or on a regular terminal: enter shell commands, run editing sessions, compile programs, etc.

Don't worry if you have already started up OPEN LOOK on your system and discovered that the screen looks nothing like this example—it won't!. We will show you how to customize your screen layout to make it look just the way you like. You will also find, if you are on a color display, that the frills around the edges of each window (called "window decorations") look more sculptured than the plain ones shown here. A nice feature of OPEN LOOK, unlike many other X11 GUIs, is that it automatically adapts to the kind of screen you are on. On a monochrome monitor, OPEN LOOK uses the "two-dimensional" look shown in most of these examples, which we've used because it prints more cleanly in the book. When running on a color or grayscale monitor, OPEN LOOK attempts to create a three-dimensional appearance[†], which is more attractive than the look provided by some older window managers. See the page of Color Plates for some examples of OPEN LOOK in color. If you're using such a monitor, you'll probably notice that window frames, various command buttons, icons, etc., appear to be raised to varying heights above screen level. This illusion is created by subtle shading and gives many display features a "beveled" look, similar to the beveled style of some mirrors.[‡]

† Purists will object to this use of "three dimensional", since it hides an entire discipline of computer graphics work in the presentation of three-dimensional images. The rest of us will content ourselves with its use.

1

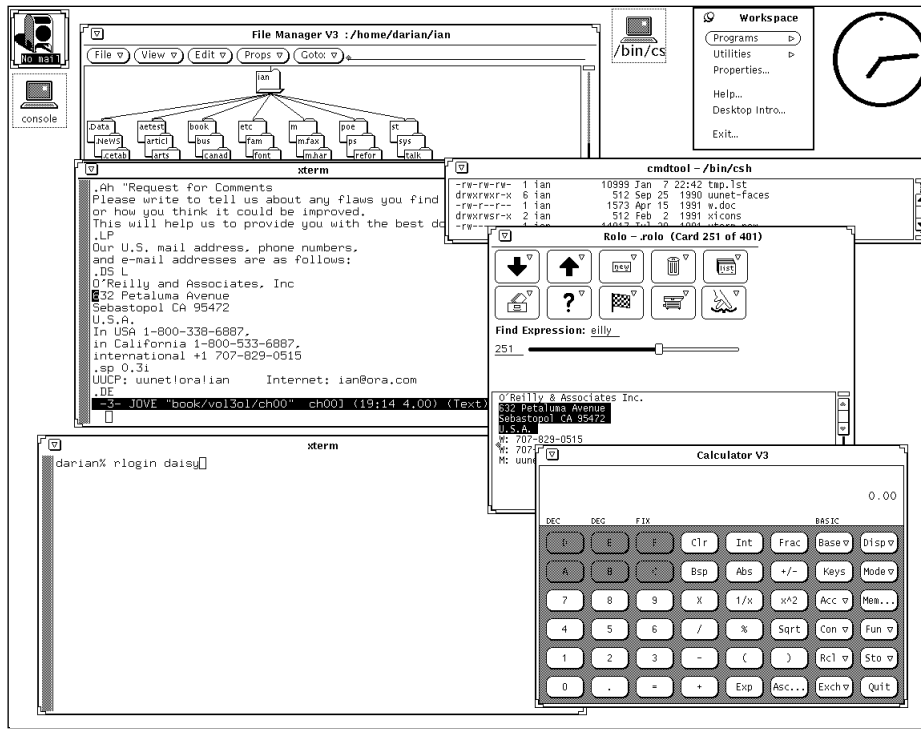


Figure 1-1. X display with numerous windows and an icon

The display in Figure 1-1 also includes other application windows: a clock, a mailing list program *xrolo*, and a calculator *calctool*. OPEN LOOK and X provide many such small utility programs—analogous to the so-called "desk accessories" of the Macintosh environment—intended to make your work easier.

The blank area that fills the unused portion(s) of the screen is conventionally called the *root* (or *background*) window; OPEN LOOK calls this area the *Workspace*. Application windows are displayed on top of this root window. X considers windows to be related to one another in a hierarchy, similar to a family tree. The root window is the root or origin within this hierarchy and is considered to be the *parent* of application windows displayed on it. Conversely, these application windows are called *children* of the root window. In Figure 1-1, the *xterm*, *cmdtool*, clock, *xrolo*, and *xcalc* windows are children of the root window.

The window hierarchy is actually much more complicated than this "two generation" model suggests. Various *parts* of application windows are windows in their own right. For example, many applications provide menus to assist the user. Technically speaking, these menus are separate windows. Knowledge of the complexity of the window hierarchy (and

‡ A few people find this "three-dimensional" look distracting, so of course OPEN LOOK provides a way to turn it off, as we'll see later. Other Graphical User Interfaces that provide a 3D look, like Motif, generally do not give you this option.

the composite parts of an application) is irrelevant at this stage, but will become important when we discuss how to tailor an application better to suit your needs.

One of the strengths of a window system such as X is that you can have several processes running in (and even writing output to) different windows simultaneously. For example, in Figure 1-1, the user is logging in to a remote system in one *xterm* window and editing a text file in another terminal emulator window. And he's fetching an address from a "Rolodex" emulator in another for use in the text file. (As we'll see in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, you can also cut and paste text between windows.) The only limitation on this multiwindow activity is that you can only be typing input into one window at a time.

Another strength of X is that it allows you to run programs on machines connected by a network. You can run a process on a remote machine while displaying the results on your own screen. You might want to access a remote machine for any number of reasons, to use a program or access information not available on your local system; to distribute the work load, etc. We'll discuss X's networking capabilities in more detail in the "X Architecture Overview" on page 20.

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

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

The window manager provided with OPEN LOOK is called *olwm*, the *O*pen *L*ook *w*indow *m*anager. There are two implementations of *olwm*, one from Sun and one from AT&T. At this stage it does not matter which you are using (only when we come to customize the window manager behavior will the differences become important). A nice feature of *olwm* is the "frame" it places around all main windows on the display. Each top-level application window in the illustration is surrounded by this frame. As we'll see, you can manage the window by clicking a mouse† or other pointing device on various parts of the window frame.

† Today, the mouse is an integral part of the computer community, but it was not always so. The first computer mouse is believed to have been invented in 1964 by Doug Engelbart at SRI, near Stanford, California. He writes (in A. Goldberg (Ed.), *A History of Personal Workstations*, p. 196): "No one is quite sure why it got named a 'mouse', or who first started using that name. None of us would have thought that the name would have stayed with it out into the world, but the thing that none of us would have believed either was how long it would take for it to find its way out there."

The *olwm* window frame is a composite of several parts, the most prominent of which are shown in Figure 1-2. The top edge of the frame is wider than the other three edges and fea-

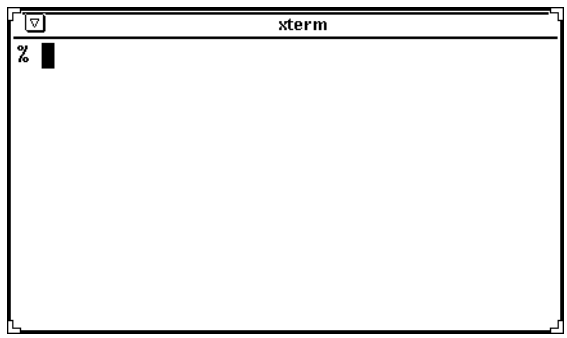


Figure 1-2. olwm frames each window on the display

tures most of the window management tools. This wide horizontal bar spanning the top of the window is called a *header*, or in standard X11 terminology, a *titlebar*. Most X11 window managers provide titlebars. The large central portion of this top edge is called the *title area* because it contains a text description of the window. (Generally this is the application name, but as we'll see later you can often specify an alternate title.) The titlebar also features a small menu mark—a box with a triangle—that we'll discuss in Chapter 6, *Using the OPEN LOOK Window Manager*. We'll also see how to use the parts of the frame to resize a window and to raise it to the top of the window stack.

olwm is intended to be used with applications built using the OPEN LOOK graphical user interface. For those not using OPEN LOOK, there are several other window managers available. In the standard distribution of X from MIT (as of X11 Release 4 and 5), the official window manager is *twm*. *twm* originally stood for "Tom's window manager," in honor of its developer, Tom LaStrange. However, it has since been worked on by many people and subsequently renamed the "tab window manager." The *twm* window manager provides a different "look and feel" than *olwm*. Rather than framing application windows, *twm* simply provides each window with a titlebar, different from the *olwm* titlebar in style, but offering similar window management functions. Other window managers exist, each with its own distinctive look and feel and its own set of features. The best known commercial offering besides *olwm* is *mwm*, which is normally used with Motif applications. However, you should note that most Motif applications will run with an OPEN LOOK window manager, and *vice versa.*

Aesthetics notwithstanding, one of the primary advantages that *olwm* shares with *mwm*, in contrast to some other window managers such as *twm*, is inherent in the nature of a frame: it provides window management tools on four sides of the window. *twm*'s titlebar is a useful window management tool, but a titlebar is often covered by other windows in the stack. In most cases at least a part of a window's frame will be visible—and thus accessible to *olwm* users but not to *twm* users.

Also pictured in Figure 1-1 are several *icon*s. An icon is a small symbol that represents a window in an inactive state. The window manager allows you to convert windows to icons and icons back to windows. You may convert a window to an icon to save space on the display, or to prevent input to that window. (Some programs, such as Sun's SunPC MS-DOS emulator, become "dormant" when iconified, and stop using CPU time.) Each icon has a label, generally the name of the program that created the window. The icons in Figure 1-1 represent another *xterm* window, a special system console window, and the file manager tool that we'll meet in Chapter 2, *Working in the OPEN LOOK Environment.* Icons can be moved around on the display, just like windows.

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

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

The cursor often changes shape as you move the pointer into different windows. A complete list of standard X and OPEN LOOK cursors is shown in Appendix D, *Standard Cursors - X11 and OPEN LOOK*. Some common MIT cursor shapes, as well as two OPEN LOOK-specific cursors, are shown in Figure 1-3. As we'll see later, some applications allow you to select the cursor to use.

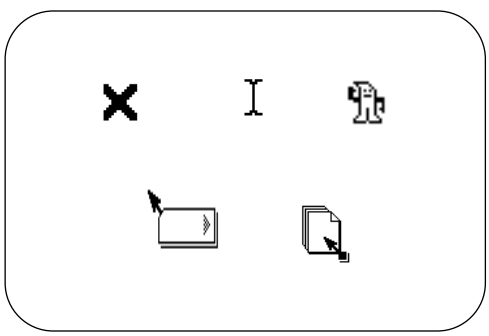


Figure 1-3. Three X11 standard cursors and two OPEN LOOK-specific cursors

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

You use the pointer to manage windows and icons, to make selections in menus, and to select the window in which you want to input. You can't type in a terminal window until you select that window using the pointer. Directing input to a particular window is called *focusing*. When a window "has the input focus," the titlebar and the text cursor (if any) are highlighted. The window to which input is directed is often called the *active* window. Most window managers require you to select the active window in one of two ways: either by moving the pointer so that it rests within the desired window or by clicking the pointer on the window. By default, the OPEN LOOK Window Manager requires you to click the pointer on the window to which you want to direct input. This focusing style is commonly referred to as "click-to-type" or "explicit focus." In this mode, the active window is indicated by darkening its titlebar, as shown in Figure 1-4.

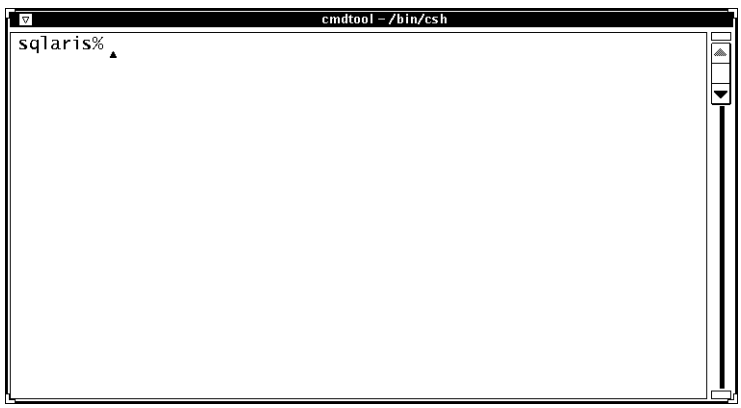


Figure 1-4. Focus on a terminal emulator window (click-to-type focus)

As we'll see in Chapter 13, *Customizing olwm*, *olwm* can be customized to allow you to direct input focus simply by moving the pointer. This alternate focusing style is commonly referred to as "real-estate-driven" (or "pointer focus"). In this mode, the active window is highlighted simply by drawing a line along the bottom of its titlebar, as shown in Figure 1-5.

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

As mentioned, the OPEN LOOK window manager uses the click-to-type focusing style by default: you must click the pointer on a window to focus input on that window. When you begin using X with *olwm*, you'll need to select the window to receive input by placing the

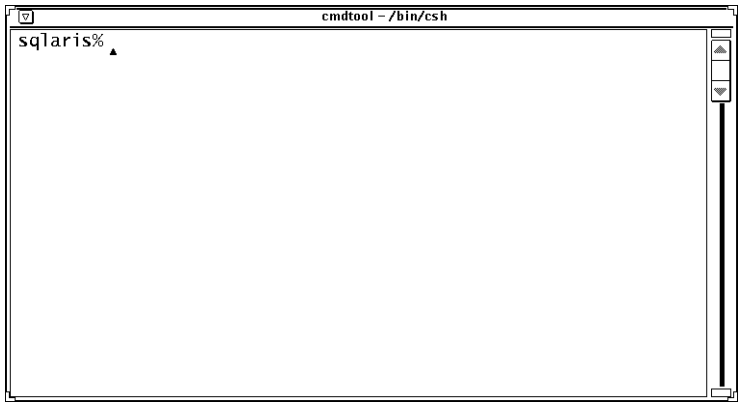


Figure 1-5. Focus on a terminal emulator window (pointer focus)

pointer anywhere within the window and clicking the button called SELECT, normally the leftmost button. †

Once you select the focus window in this way, all input is directed to that window until you move the pointer and deliberately click on another window. If you click SELECT on the root, then no window has the input focus until you click SELECT on the titlebar of a window.

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

We find click-to-type focus somewhat laborious. However, a touch typist, who is not inclined to look at the screen, might consider pointer focus too risky. It's possible to knock the pointer out of a window and type a large amount of text into the wrong window before noticing. Another disadvantage of pointer focus is that it sometimes takes a moment for the input focus to catch up with the pointer, especially on slower machines. If you type right away, some keystrokes may end up in the window you left rather than in the new window. This problem is not unique to *olwm*, but rather is a side effect of the additional overhead caused by the generality (specified in a document called the ICCCM; see Volume Zero, *X Protocol Reference Manual*) involved in complex window managers, and can affect any ICCCM-compliant window manager such as *olwm*, *mwm*, or *twm*. Since you can change the focus policy rather simply, even while the OpenWindows version of *olwm* is run-

† The middle button is called ADJUST, and the right button is called MENU. To remember which button is which, think of "SAM" for (in order) SELECT, ADJUST, and MENU. We'll see how to use all these buttons as we progress. If you are left handed and want to reverse the position of these buttons, see the last section of Chapter 14, *Customization Clients*.

ning†you might want to experiment with both methods. For now, we'll assume you're using the default click-to-type focus.

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

The pointer is also often used to display menus. Most X programs, notably *olwm*, *shelltool* and *xterm*, have menus that are displayed by keystrokes and/or pointer button motions. *olwm* provides two default menus—the *Window Menu* and the *Workspace Menu*—each representing a different menu "style."

The *Window Menu* is a "pull-down" menu that can be displayed on any window by placing the pointer on the titlebar on top of the frame and either clicking or pressing and holding down the MENU pointer button. Roughly defined, a pull-down menu is accessed from a graphical element that is always displayed, such as a command button, a menu bar, or a titlebar. Figure 1-6 shows a terminal emulator window (in this case *cmdtool*) with the *Window Menu*) displayed by clicking the MENU pointer button in the menu button on the frame.

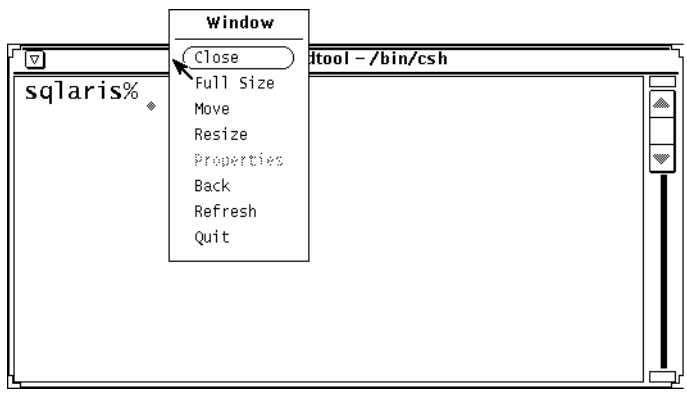


Figure 1-6. A pull-down menu: olwm's Window Menu

As you might infer from some of the menu items, you can use the *Window Menu* to hide, resize, and otherwise manage the window on which it is displayed. When you display the *Window Menu* by clicking the MENU pointer button (as opposed to pressing and holding it down), the default item (the one you would get just by clicking SELECT on the triangle mark) is surrounded by an oval box. In this case, the first available selection is *Close*, used to change a window into an icon. Of course, the first item in the *Window Menu* of an icon

† You can change it at start-up time by specifying **`-follow`** on the *olwm* command line, by setting the X Resource OpenWindows.SetInput: to followmouse, or change it at will by putting an item with a call to the FLIP-FOCUS built-in function in your *.openwin-menu.* file, and clicking on it. All these methods are described in more detail in Chapter 13, *Customizing olwm*.

is *Open* which converts the icon into a window. The third item, *Properties*, is not available yet; it is meant to provide access to a set of Window Properties. The fact that it is not selectable is indicated by the fact that it appears in a lighter (or stippled) typeface. The second item, *Full Size*, and other aspects of the *Window Menu*, are discussed in detail in Chapter 6, *Using the OPEN LOOK Window Manager*.

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

In Figure 1-6, the arrow next to the menu title represents the pointer. As you drag the pointer down the menu, each of the menu selections is highlighted (OPEN LOOK programs generally highlight an item by placing an oval box of reverse video around it). When you release the pointer button, the highlighted menu item is selected. If you drag the pointer right off an OPEN LOOK menu without selecting anything, the menu stays up until either you release the pointer button (in which case nothing is selected and the menu pops down), or you move the pointer back into the menu and select something.

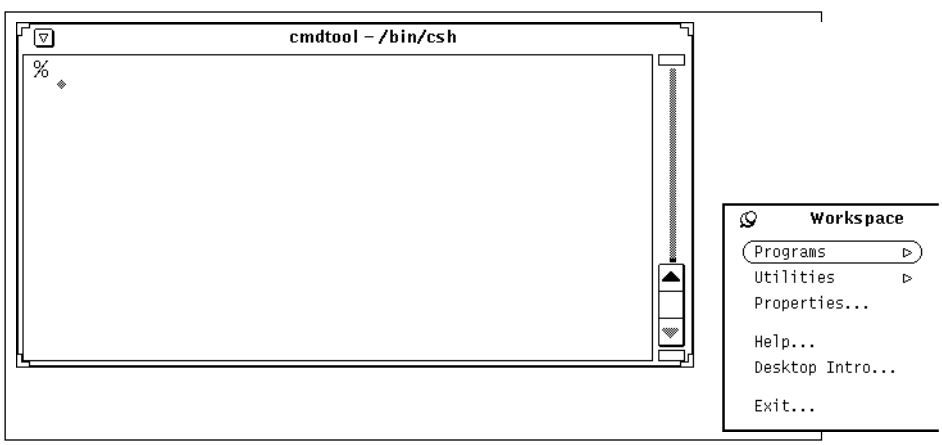


Figure 1-7. A pop-up menu: olwm's Workspace (Root) Menu

The *Workspace* or *Root Menu* (see Figure 1-7) provides commands that can be thought of as affecting the entire display (as opposed to a single window). For example, the first menu item, *Command Tool* (on OpenWindows) creates a new *cmdtool* window on the local machine and display.

Though *olwm*'s menus are useful, you'll probably find that you perform many window management functions simply by using the pointer on the window frame. In Chapter 6, *Using the OPEN LOOK Window Manager*, we'll describe several of these functions. Keep in mind, however, that both of the menus can be useful in certain circumstances. For

instance, the *Window Menu* may be useful when parts of the window frame are obscured by another window. The *Workspace Menu* can be customized to execute system commands, such as the terminal emulator command initialized by the *Command Tool*'s item under *Programs*. It's fairly simple to add items to the OPEN LOOK root menu or *Workspace Menu* for the applications you use regularly. Just to show you the range of flexibility, Table 1-1 shows an alternative workspace menu that we use.

Table 1-1. Alternate Workspace Menu

Name	Function
Local Windows	Create local terminal emulator windows
LANlogins	Terminal emulator windows on our Local Area Network
Internet	Terminal emulator windows on Internet hosts
Window Programs	One of several dozen X- and NeWS- based applications
OpenWin Demos	A link to the standard OpenWindows demo menu file
Lock Screen	A standard X11 screen lock program
Terminations	Shutdown X or even shutdown my workstation

Chapter 13, *Customizing olwm*, describes how to add menu items and other modifications.

As we'll see in Chapter 8, *Other Standard Clients*, a few non-OPEN LOOK programs provide "jump-up" menus that you display simply by placing the pointer on a particular part of the window, e.g., a horizontal menu bar across the top.

A final note about the X display: in X, the terms *display* and *screen* are not equivalent. A display may consist of more than one *screen*. This feature might be implemented in several ways. There might be two physical monitors, linked to form a single display, as shown in Figure 1-8. Alternatively, two screens might be defined as different ways of using the same physical monitor. For example, on certain older Sun Workstations (any Sun with a "cgfour" color display), screen 0 is normally color, and screen 1 is black and white.†

† With OpenWindows, you must specify the screens to use and their order with the `-device` command line option; the default is a single screen in color. See the example in the OpenWindows section of Chapter 2, *Working in the OPEN LOOK Environment*, or the *openwin* manual page in Part III of this guide for details. The MIT "Xsun" server uses the same technique on a cgfour, but automatically uses both screens. With either server, the OpenWindows version of *olwm* automatically finds all the screens that are in use.

Each screen is the size of the monitor; you can only view one screen at a time. In practice, the two screens seem to be side by side: you can "scroll" between them by moving the pointer off either horizontal edge of the screen. By default, windows are always placed on screen 0 but you can place a client window on screen 1 by specifying the screen number in the `-display` option when starting the client. (See Chapter 2, *Working in the OPEN LOOK Environment*, for instructions on using the `-display` option.)

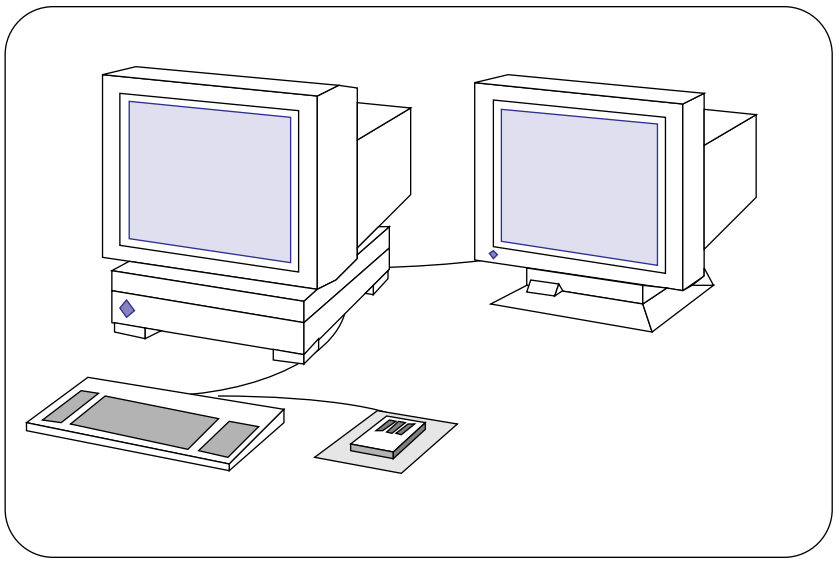


Figure 1-8. A display made up of two physical screens

1.2 Standard X Clients, OPEN LOOK Clients and Motif Clients

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

However, the look and feel of an application is not entirely determined by the window manager. The programming routines used to create the application also distinguish it. The *olwm* and *cmdtool/shelltool* programs were written with an OPEN LOOK-specific toolkit, while the *xterm* program was been written using "the X Toolkit", developed at MIT.

The X Toolkit is a collective name for two subroutine libraries designed to simplify the development of X Window System applications: the X Toolkit (Xt) Intrinsics and the Athena widget set (Xaw). The Xt library consists of low-level C programming routines for building and using *widgets*, which are pre-defined user interface components or objects. Typical widgets create graphical features such as menus, command buttons, dialog boxes,

and scrollbars. Widgets make it easier to create realistic applications. A common widget set also ensures a consistent user interface between applications.

As mentioned, most of the "standard" MIT X applications are written using the X Toolkit, and have the user interface provided by the Athena Widget set.

One of the first implementations of OPEN LOOK for X Windows is AT&T's OPEN LOOK Intrinsics Toolkit (OLIT). As the name implies, OLIT is based on the Xt Intrinsics, and consists primarily of a widget set that implements the OPEN LOOK GUI. An application coded using the OLIT Toolkit, or any other OPEN LOOK toolkit, has a particular look and feel that is prescribed in the *OPEN LOOK Style Guide* and illustrated by most of the pictures in this book.

Another OPEN LOOK implementation for X is the XView toolkit, designed by Sun to allow the thousands of SunView applications to migrate to OPEN LOOK and The X Window System. Its programmer interface is similar to that of the older proprietary SunView window system, but it is an OPEN LOOK toolkit that works with X.

A third OPEN LOOK toolkit—one that is being phased out in favor of the others—is "The NeWS Toolkit" (TNT), also from Sun. This toolkit is neither Xt- nor X11-based, but uses the alternate NeWS protocol to communicate with the display server. It is included in Sun's OpenWindows package; applications written using TNT should behave much the same as those developed using either of the other OPEN LOOK toolkits.

The Open Software Foundation has also developed an Intrinsics-based widget set – or more precisely, a specification for a GUI—called Motif. There is only one sanctioned implementation of the Motif specification, and if you want it, you must license it from the Open Software Foundation. OPEN LOOK and Motif are the prime contenders to establish a graphical user interface standard in the market. In fact, they *are* the two standard GUIs in the workstation market. We will discuss OPEN LOOK clients throughout most of this book. Since you may occasionally need to use Motif-based clients, we show one sample Motif program later in this chapter, and summarize the important aspects of using Motif programs in Appendix M, *OPEN LOOK and Motif*.

Some of the clients discussed in this guide are standard X clients shipped by MIT, while others are part of the commercial OPEN LOOK offerings. Most of the MIT clients have been built with the X Toolkit and illustrate the use of many of the Athena widgets. When you run these standard clients (or any Motif clients) with the *olwm* window manager, your environment is something of a hybrid—neither a vanilla X nor a pure OPEN LOOK environment.

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

Without dissecting every component or closely examining how it functions, let's briefly compare a standard X application to an OPEN LOOK application, highlighting some of the major differences (primarily in appearance). Many features of OPEN LOOK and standard X

applications also *operate* differently. We'll examine the functionality of various OPEN LOOK and Athena widgets in more detail in Chapter 8, *Other Standard Clients*.

For the comparison, we'll use an OPEN LOOK client program called *textedit*, the OpenWindows standard window-based text editor. *textedit* (see Figure 1-9) demonstrates many of the features of OPEN LOOK as a GUI. It also demonstrates most of the OPEN LOOK-specific editing actions: editing text in this program is the same as editing commands in *cmdtool*'s history mechanism, for example. This aspect of *textedit* is described more fully in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

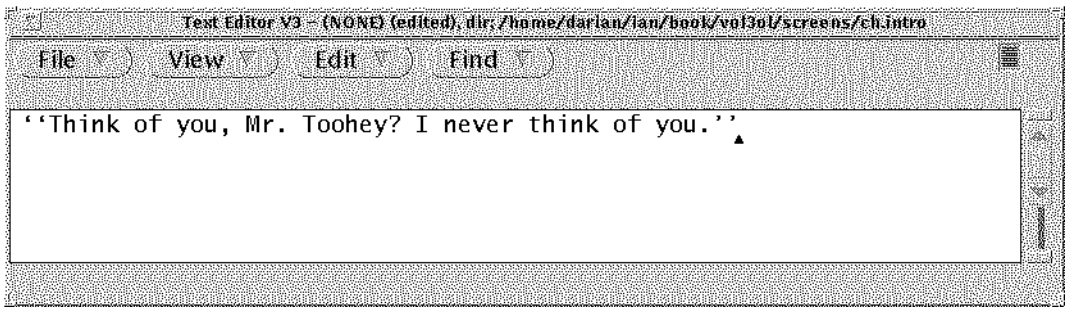


Figure 1-9. An OPEN LOOK application: textedit

The standard X client we're using is *xclipboard* (Figure 1-10), which you use in concert with *xterm*'s "cut and paste" facility, described in Appendix A, *The xterm/olterm Terminal Emulator*. The *xclipboard* client provides a window in which you can paste multiple text selections and from which you can copy text selections to other windows. Similar to the clipboard feature of the Macintosh operating system, *xclipboard* is basically a storehouse for text you may want to paste into other windows.

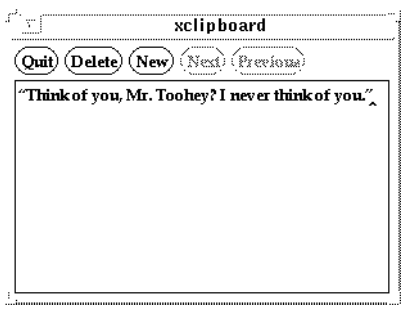


Figure 1-10. An MIT X application: xclipboard

A Motif application displayed in Figure 1-11 is *mre*, the Motif-based Resource Editor (we'll talk about Resources themselves later). *mre* is a demonstration program that is normally shipped with Motif, but isn't part of the official Motif standard. In fact, the Motif

standard doesn't document any standard clients, so your friend's Motif system down the hall may or may not include it.

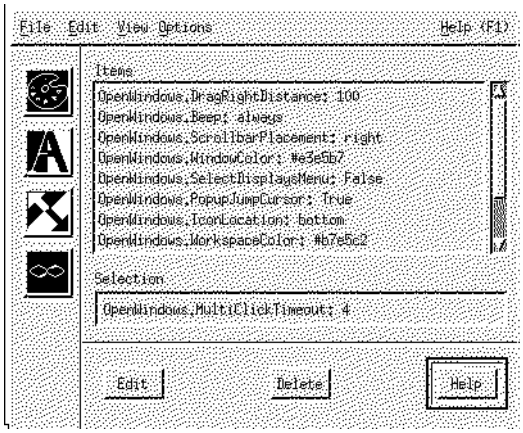


Figure 1-11. A Motif application: mre

The most striking difference between the first two clients is simply the amount of detail. Like the OPEN LOOK window manager, the individual OPEN LOOK controls create a "three-dimensional" appearance, at least on a color or grayscale monitor†. The push buttons are shaded to suggest that they are raised above the level of the application window. The menu bar is shaded to appear raised. The scrollbars have clearly distinguishable components, all of which are shaded and contoured to maintain the 3-D impression. For a better look at this effect, see the page of color plates; it has been reduced to monochrome for printing here. The Motif window, too, uses a bevelled or 3-D impression.

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

Each application has a series of buttons along the top. In *xclipboard*, each button simply does one action. In an OPEN LOOK or Motif application, however, these buttons are usually pull-down menus. OPEN LOOK clearly indicates these menus by the downwards-pointing triangle. *File*, *View*, and *Edit* are common on OPEN LOOK applications. You can activate one of these menus by placing the pointer on it and clicking (or pressing and holding down) the MENU (right) pointer button. Alternately, you can select the default action from that menu, without making the menu appear, by clicking the SELECT (left) pointer button while the pointer is in the button. The *xclipboard* application doesn't provide any menus—it's a fairly simple program. However, some standard X clients (notably *xman* dis-

† Most of the screenshots of OPEN LOOK applications in this book are monochrome, and have a "two-dimensional" look. The *textedit* application was captured in grayscale to show the bevelling effects.

cussed in Chapter 8, *Other Standard Clients*) provide pull-down menus accessed by pressing and holding down a pointer button.

OPEN LOOK pull-down menus have a few advantages over pull-down menus provided by standard MIT X clients. While you must press and hold down a pointer button to display a menu provided by a standard X client, you can display an OPEN LOOK menu simply by clicking a pointer button—and the menu stays displayed until you click again. OPEN LOOK menu items can also be invoked in multiple ways (including pointer actions and keystrokes); the only way to invoke an item from a standard X menu is by dragging the pointer down the menu and releasing the button. The various ways you can work with an OPEN LOOK pull-down menu are described in Chapter 2, *Working in the OPEN LOOK Environment*.

One minor advantage of Motif applications is that their menu bar features *sliding menus*; if you hold the left button and slide the pointer across the menu bar, each menu will pop down without having to be clicked on.

Despite differences in general appearance and complexity, *textedit* and *xclipboard* have many analogous components. Both applications feature a subwindow containing text that can be edited.

Each application features buttons: push buttons in the *textedit* window; command buttons in the *xclipboard* window, drawn buttons in the *mre* window. From a user's viewpoint, push buttons and command buttons are functionally equivalent (though you can invoke a push button's function in more ways). OPEN LOOK and the Athena widget set simply identify them by different names. This Motif application, like some OPEN LOOK applications, features drawn or "glyph buttons," buttons decorated with a small picture instead of being labeled with a word or two of text.

All three application windows have a vertical scrollbar, used to display text that is currently outside the viewing window. (These scrollbars are only displayed when the text read into the window extends beyond the bounds of the viewing area. If the text only exceeds the viewing area in one direction—either horizontally or vertically—only one scrollbar will be displayed.) The Athena scrollbar is basically rectangular (actually one rectangle within another). The OPEN LOOK scrollbar is much more intuitive—it provides the visual analogy of an elevator used in a multi-story building. Notice the arrows on either end of the indicator mark in the middle, for instance. These arrows are the hallmark of an OPEN LOOK scrollbar and can help you readily identify an OPEN LOOK application. The arrows also provide functionality not duplicated by the Athena scrollbar. Similarly, the Motif scrollbar can be used to recognize Motif applications.

In general, once you've mastered the basics of working with OPEN LOOK client programs running under the *olwm* window manager, you should have no problem making use of any additional features provided by commercial applications built with OPEN LOOK or any similar GUI.

1.3 X Architecture Overview

Before X11 became popular, most window systems were *kernel-based* and *host-based*: that is, they were closely tied to the operating system itself and could only run on a discrete system, such as a single workstation. X, by contrast, is a *user-mode*, *network-based* window system; it is not part of any operating system but instead is composed entirely of user-level programs.

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

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

This division within the X architecture allows the clients and the display server either to work together on the same machine or to reside on different machines (possibly of different types, with different operating systems, etc.) that are connected by a network. For example, you might use a relatively low-powered PC or workstation as a display server to interact with clients that are running on a more powerful remote system. Even though the client program is actually running on the more powerful system, all user input and displayed output occur on the PC or workstation server and are communicated across the network using the X protocol. Figure 1-12 shows a diagram of such a network.

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

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

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

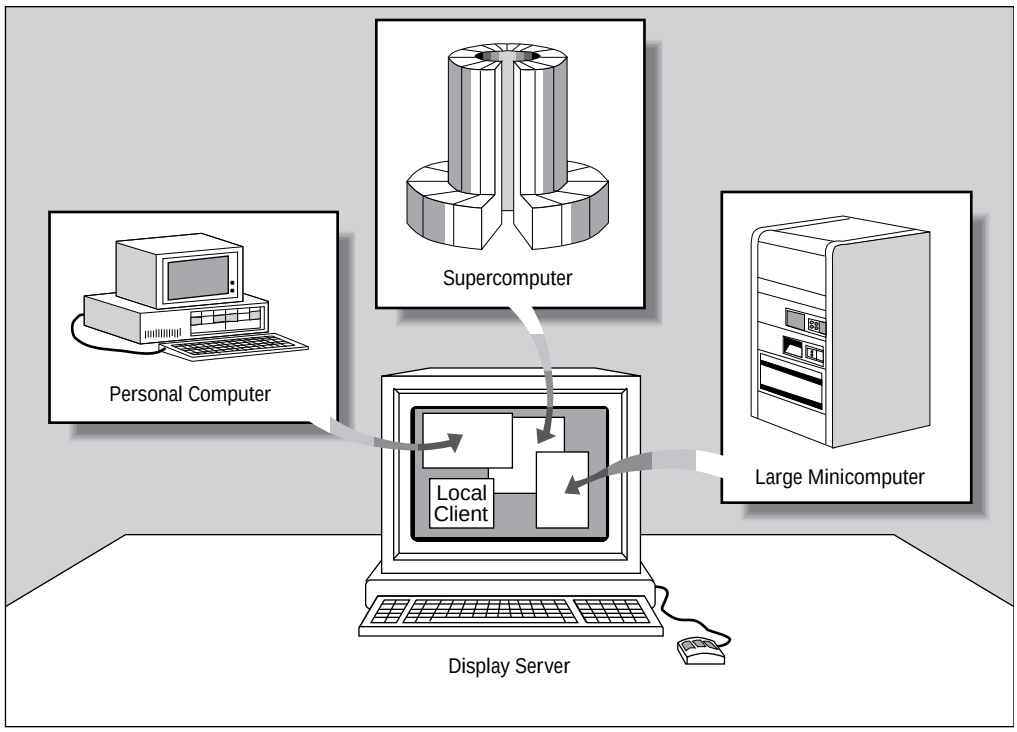


Figure 1-12. A sample X Window System configuration

1.4 The X Display Server

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

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

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

Servers are available for PCs with 80386 processors, for workstations, and even for special terminals (called X terminals), which may have the server downloaded from another machine or stored in "read-only memory" (ROM). Details on the variety of servers available is included in Volume Eight, *X Window System Administrator's Guide*.

The server and the client programs communicate using a rigidly—defined protocol, The X Protocol, described in Volume Zero, *X Protocol Reference Manual*. Some servers may recognize particular "extensions" to the base protocol.

The Sun OpenWindows server uses the standard X Protocol, but for increased flexibility OpenWindows Versions 2 through 3.1 also accept connections using another, lower overhead protocol called the NeWS (Network-extensible Window System) protocol. The NeWS language allows the application programmer to extend the environment by writing procedures in the PostScript language. NeWS clients will only work with a NeWS server, which means an OpenWindows server; they will be discussed in this book with the notation "OpenWindows Only".

Interestingly, some of the features that have become "X extensions" were first featured as standard parts of NeWS. These include the SHAPE extension (used to draw circular windows) and the DISPLAY POSTSCRIPT extension. Display Postscript (DPS for short) is less flexible than the NeWS protocol, but does give programmers who must use X-based toolkits the ability to use the PostScript language. DPS has the further advantage that DPS clients will work with a DPS server from any vendor. Neither OpenWindows 2 nor 3.1 used DPS; the NeWS protocol in the server is Sun's original display-based PostScript interpreter. However, Sun and Adobe (the creator of Display Postscript) signed a technology transfer agreement in late 1992 which will see the OpenWindows product switching over to Display PostScript soon †

1.5 Clients

As previously mentioned, a client is an application program. The OPEN LOOK Window Manager, *shelltool*, *xterm*, and *calctool* are client programs we've seen already. The standard release of X from MIT includes more than 50 client programs that perform a wide variety of tasks. Another few hundred client programs are available as "contributed" or "free software," and more are added each week. X allows you to run many clients simultaneously: each client displays in a separate window. For example, you could be editing a text file in one window, compiling a program source file in a second window, and reading your mail in a third, while displaying the system load average in a fourth window.

While X clients generally display their results and take input from a single display server, they may each be running on a different computer on the network. It is important to note that the same programs may not look and act the same on different servers: users can run different window managers on different servers; users can customize X clients differently on each server; and the display hardware on each server may be different.

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

† This will happen by mid-1993?, probably with Release 3.2? of the OpenWindows system.

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

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

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

! Note that OPEN LOOK has an additional, unrelated meaning for the term "properties", referring to "the properties of an application." This use of the term means "the user-controllable aspects of that program's behavior." This is exemplified by "property sheets" that let you edit the properties of an application; most OPEN LOOK applications include a Properties menu item, and the root menu has a like-named item that lets you edit global settings by adjusting X Resources. This is described in Chapter 6, *Using the OPEN LOOK Window Manager.*

We've already discussed one important client, the OPEN LOOK Window Manager. Several other frequently used client programs are discussed in the following sections.

1.5.1 The ShellTool terminal emulator

X11 itself is designed to support only bitmapped graphics displays. But there are many day-to-day operations that are best handled by a command-line interpreter like the UNIX shells. For this reason, one of the most important clients is a *terminal emulator*. The terminal emulator brings up a window that allows you to log in to a multiuser system and to run applications designed for use on a standard alphanumeric terminal. Anything you can do on a terminal, you can do in this window. If you are using Sun's OpenWindows package, the terminal emulator used in the default setup is called *shelltool* or *cmdtool* (the two names refer to a single program under two different modes).

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

1.5.2 The xterm Terminal Emulator

There are several terminal emulators besides *cmdtool*. Users of MIT and SVR4 X systems generally use one called *xterm*. *xterm* is perhaps the most widely available terminal emulator, and it is available under OpenWindows too. *xterm* emulates a character-based terminal or a vector-graphics terminal. The standard MIT X11 version of *xterm* emulates a DEC VT102 character terminal; the *xterm* shipped with AT&T-OL emulates an AT&T 6386 console in character mode. Both versions can also emulate a Tektronix® 4014 as their graphics terminal. For each *xterm* process, you can display both types of windows at the same time but only one is active (i.e., responding to input) at a time. This is one area where *xterm* would seem to excel over Sun's *cmdtool*, since the latter has no ability to emulate a Tektronix terminal.†

1.5.3 Other X Clients

The standard distribution of X from MIT includes more than 50 client applications; most of these are included in OpenWindows and AT&T-OL. The client you will probably use most frequently is a terminal emulator, either *shelltool* or *xterm*. We've grouped some of the other more useful applications as follows: the "Availability" column lists "Standard" (standard MIT clients), "Contributed" (free software, obtain from `Uunet` or MIT, see the *Preface*), "OPEN LOOK" for clients that should be in any OPEN LOOK system, or "OpenWindows" for clients that are only included with Sun's Open Windows.

† Since the ANSI and Tektronix terminals are so different, Sun's older SunView window system featured a separate program called *tektool*, for that purpose. However *tektool*, has not (yet) been brought forward to OPEN LOOK. If you are an OpenWindows user in need of Tektronix emulation, the SunView version is still included in SunOS 4.1, and it can be used under OpenWindows, subject to the limits laid out in Appendix L, *Running SunView Applications on OpenWindows*. Or, you could use *xterm* for the times when you need Tektronix mode.

These and other client applications are described in Chapters 5 through 9. In addition, a ref-

Table 1-2. An Applications Sampler

Name	Description	Availability
Desktop Manager		
filemgr, olfm	File manager (desktop manager)	any OPEN LOOK implementation
Desk accessories		
xbiff	Mail notification program	Standard
faces	Fancier, visual mail notifier	Contributed
xclock, oclock	Clock applications	Standard
clock	Fancier clock	OpenWindows
xcalc	Desktop calculator	Standard
xload	System load monitor	Standard
perfmeter	Fancier load monitor	OpenWindows and xview only
xman	Manual page browser	Standard
xrolo	XView Rolodex™ emulator.	Contributed
cm	Calendar Manager	OpenWindows
calentool	Calendar manager	Contributed
Display and keyboard preferences		
properties	Sets display, keyboard and color preferences; updates your resource file accordingly	Any OPEN LOOK implementation
xset	Sets display and keyboard preferences, such as bell volume, cursor acceleration, and screen saver operation	Standard.
xmodmap	Allows you to map keyboard keys and pointer buttons to particular functions	Standard
Text Editing utilities		

Table 1-2. An Applications Sampler

Name	Description	Availability
xedit	Simple text editor	Standard
textedit	Simple text editor	OpenWindows
Font utilities		
xlsfonts	Lists available fonts	Standard
xfd	Displays the characters in a single font	Standard
xfontsel	Allows you to display multiple fonts sequentially and select a font to be used by another application	Standard
Graphics utilities		
bitmap	Bitmap editor	Standard
iconedit	Another bitmap editor	OpenWindows
atobm, bmtoa	Convert ASCII characters to bitmaps and bitmaps to ASCII characters	Standard
pageview	Show PostScript output	OpenWindows
snapshot	Fancy screen capture program	OpenWindows
xloadimage, xv	Control, display and manipulate bitmap files in a variety of formats	Contributed
xfig, xpic	One of many drawing programs. Also touchup,.others	Contributed
Printing applications		
xwd, xwud	Dumps and re-displays the image of a window to a file	Standard
xpr	Translates an image file produced by *xwd* to PostScript® or another format, suitable for printing on a variety of printers	Standard
xdpr	Combines xwd, xpr and lp/lpr	Standard
Removing a window		

Table 1-2. An Applications Sampler

Name	Description	Availability
xkill	Terminates a client application	Standard
pam	Hides a dead window	OpenWindows only
Window and display information utilities		
xlsclients	Lists the clients running on the display	Standard
xdpyinfo	Lists general characteristics of the display	Standard
xwininfo	Lists general characteristics of a selected window	Standard
xprop	Lists the properties associated with a window	Standard
psps	Lists the PostScript (and other) processes (window programs) in the XNeWs server	OpenWindows only

erence page describing each client and listing its options appears in Part Three of this guide. As more commercial and user-contributed software is developed, many more specialized programs will become available. For example, at the time of this writing (mid-1992), the word processing user has the choice of many X- or OPEN LOOK-based word-processing programs, including Word Perfect, FrameMaker, Interleaf, IslandWrite, DECWrite, and others, with SoftQuad Author/Editor and ArborText Publisher for more involved electronic publishing projects. The desktop graphics market includes Arts & Letters, Corel Draw, Island Draw/Paint, and others. Just about any type of commercial application you could want is now available for The X Window System, and most of them are available for the OPEN LOOK GUI environment. A list of programs—both commercial and contributed—that use the OPEN LOOK GUI is published in the Usenet newsgroup *comp.windows.open-look*, and a recent version of this is reprinted in XXX NEW APPN.

1.5.4 Customizing Clients

Most X clients are designed to be customized by the user. A multitude of command-line options can be used to affect the appearance and operation of a single client process. A few of the more useful command-line options are introduced in Chapter 2, *Working in the OPEN LOOK Environment.* Chapter 11, *Command-line Options*, discusses several options in detail. Part Three of this guide includes a reference page for each client that details all valid options.

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

Generally, these resource values are loaded into the server using a program called *xrdb* (*X r*esource *d*ata*b*ase manager). Then the values are accessed automatically when you run a client. Storing your preferences in the server with *xrdb* also allows you to run clients on multiple machines without maintaining an *.Xresources* file on each machine. Because editing these resources by hand can be tedious, OPEN LOOK specifies a program to manage the resources that affect global OPEN LOOK characteristics. On OpenWindows this program is called *props* (for "properties"); on AT&T-OL it is called *olwsm*, the "workspace manager." It can be started by name or from the root menu. We will see this program in detail in Chapter 6, *Using the OPEN LOOK Window Manager*.

In addition, many OPEN LOOK applications have a "properties" menu that will allow you to edit the resources (or other customization file) for that application. We'll see examples of these in various places.

There is a separate customization file for the *olwm* window manager (called *.openwin-menu* on OpenWindows or *.olprograms* in AT&T-OL) which is also kept in your home directory. On OpenWindows, editing the *.openwin-menu* file lets you modify the root menu (the *Workspace* menu). On AT&T-OL, you can only change the list of programs to be run from the *Programs* sub-window of the *Workspace menu*; you use the *properties* program to modify this list of programs. Unlike some other window managers, *olwm* does not let you modify the behavior of the window manager, nor the key and pointer button sequences used to invoke actions, by editing this file. Instead, you use "X Resources", most of which can be set by the *properties* program. Chapter 12, *Setting Resources*, has more information.

Client customization is introduced in Chapter 14, *Customization Clients*, and is the general topic of Part Two of this guide.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 2

Working in the OPEN LOOK Environment

The only way to learn any complex task is by doing it. In the first part of this chapter we show you how to start up an X Window System session, including the X server and at least one terminal window, along with the *olwm* window manager. The chapter then guides you through some operations that are common to all programs using the OPEN LOOK GUI, as well as to most X11 programs. This chapter describes how to:

- Start your X system, if it isn't started automatically
- Use the mouse or other pointer device to choose a window
- Use the pointer to move, reshape, iconify, and even delete windows
- Understand and use all the standard OPEN LOOK controls
- Start additional client programs, on both local and remote machines

This chapter is tutorial, and can be worked through at a computer with a version of the OPEN LOOK GUI software installed.

2.1 Getting Started with X and OPEN LOOK

There are several possible ways you can start X. If you are lucky, your system administrator will have read Volume Eight, *X Window System Administrator's Guide*, and configured your system to use the X Display Manager (*xdm*) login program as shown in Figure 2-1. If this is the case, you need only type your normal login and password to begin using X.

Or, you may have available a comprehensive vendor-supplied X package such as Sun's OpenWindows, in which case you need only log in at the workstation console (see Figure 2-2), and type one command, normally *openwin*.

On the other hand, you may be using a dedicated terminal called an X Terminal. Each of these methods is described in the following sections. If you don't fit into one of these cat-

Figure 2-1. An XDM Screen

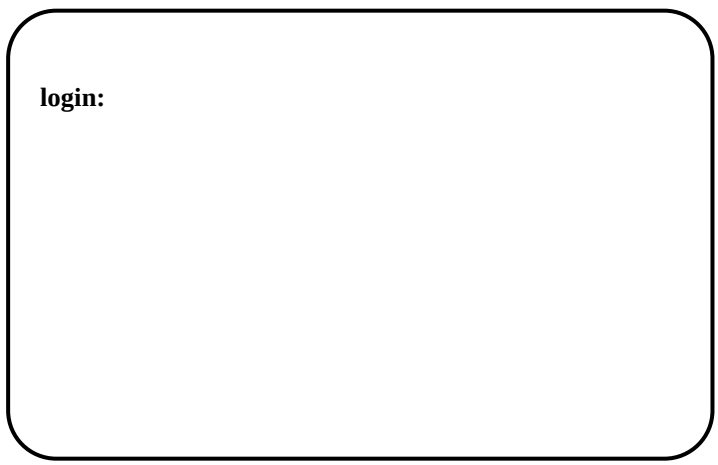


Figure 2-2. A Console Login

egories, you are probably on your own, and should consider reading Volume Eight, *X Window System Administrator's Guide*.

2.1.1 Logging In via the Special xdm Window

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

Log in by typing your user name and password at the prompt, pressing RETURN after each. Without any user customization, the display manager executes a standard login "session," providing the first *xterm* window and starting the window manager. The *xterm* window will be displayed in the upper-left corner of the screen. If the OPEN LOOK window man-

ager is running, you will see the characteristic frame around the *xterm* window, as in Figure 2-3. The name of the application ("xterm") appears in the frame's title area.

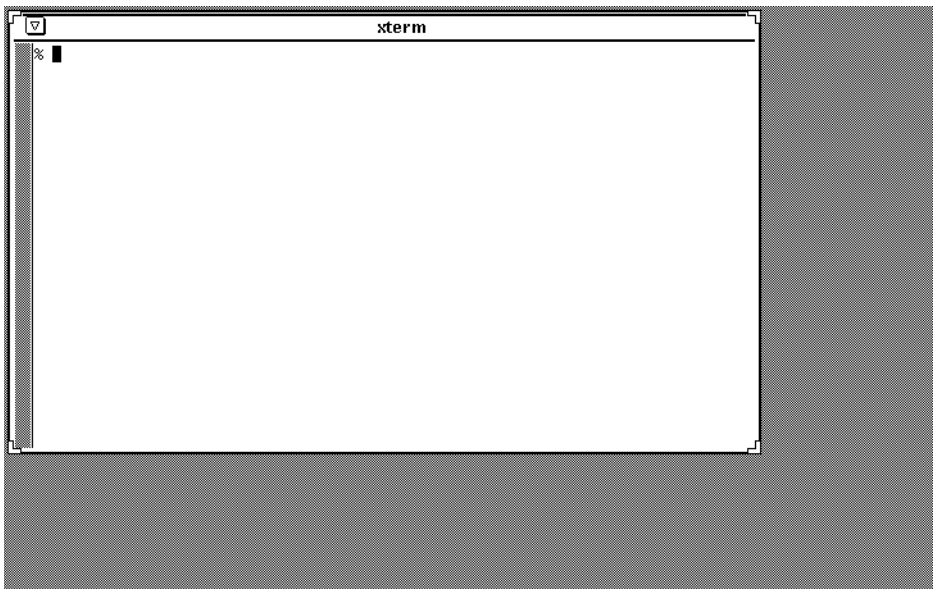


Figure 2-3. Window frame indicates that olwm is running

The frame provides a quick and easy way to move, iconify, resize, and otherwise manage windows on the screen. Some of the basic window manager functions are described below and in Chapter 6, *Using the OPEN LOOK Window Manager*.

You can now proceed to the section "Selecting and Using a Window" on page 35.

2.1.2 Logging In at a Full Screen Prompt: Starting OpenWindows or X

If you want to use a workstation that is not displaying any windows, but instead has a simple prompt ending with "*login:*", you just need to log in and start X manually. Typically, when you log in, no windows are opened; instead the entire screen functions as a single terminal, as shown in Figure 2-2.

If your account was created with the Sun-provided6 startup scripts (for example using SunOS 4.1.x's *adduser* script), then you will get this message

```
Starting OpenWindows, Control/C to interrupt...
```

and OpenWindows will be started for you automatically. If not, maybe your system administrator will be persuaded to configure *xdm* for you...

If no windows are displayed when you log in at a full screen prompt, you can start the X Window System manually. In this section we'll assume that an OPEN LOOK package such as Sun's OpenWindows has been installed. You need only type the command

```
% openwin
```

and you should (after a delay) see the full screen change color. Then an OpenWindows logo similar to Figure 2-4 should appear. If instead, you get a message like "command not found", then you may need to specify the full UNIX path to the start-up script, which is usually

```
% /usr/openwin/bin/openwin
```

The resulting screen should look like Figure 2-4.

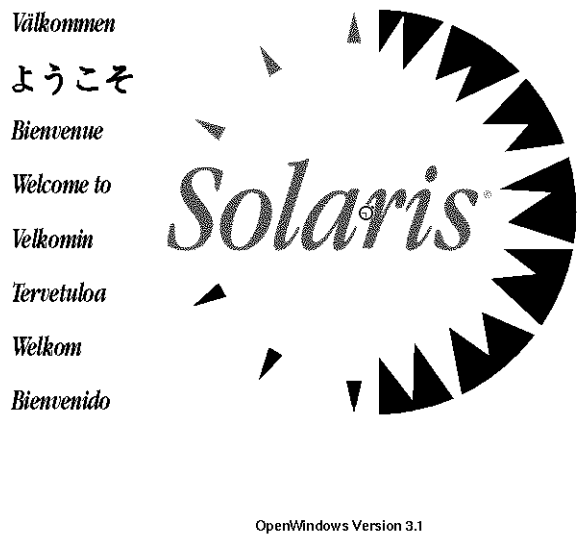


Figure 2-4. Sun OpenWindows startup screen

If you have an older version of OpenWindows then the welcome screen will be simpler, with a logo like that of Figure 2-5 in the center of the screen.

After some time (depending on your workstation), the screen will clear again, and you will get some default clients like those in Figure 2-6.

You can use the optional `-dev` argument to specify a non-default "frame buffer" device—the controller for a graphics monitor connected to your workstation (normally the

Figure 2-5. Older OpenWindows startup screen

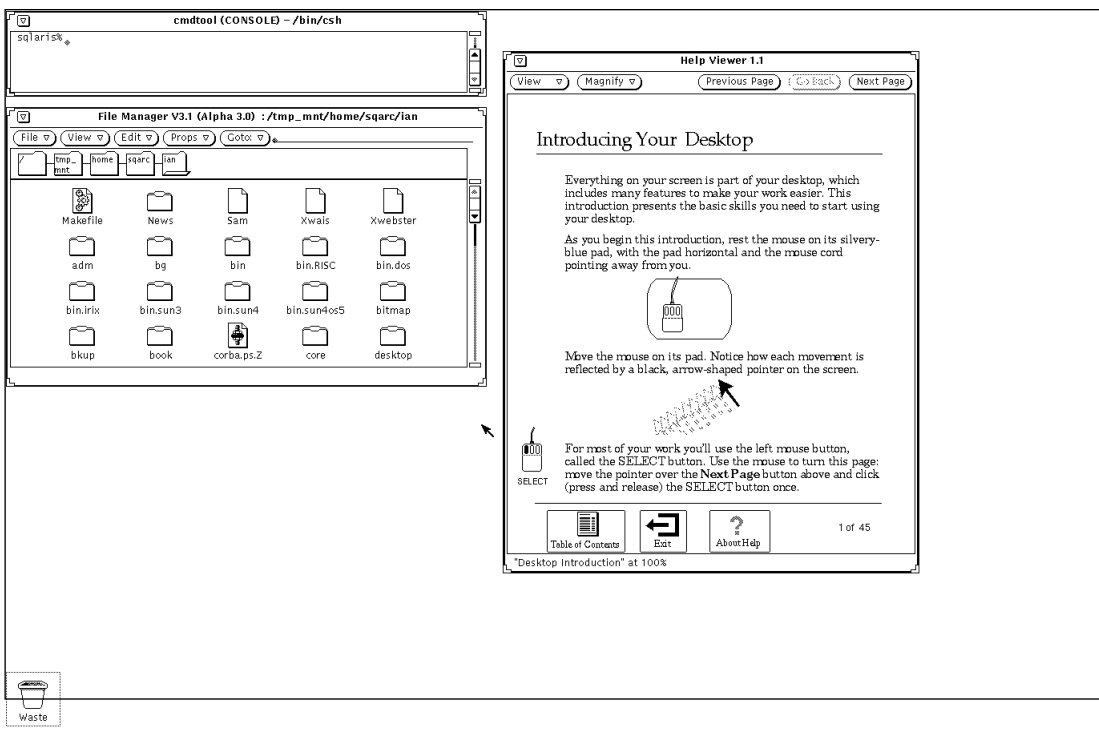


Figure 2-6. OpenWindows initial screen—not customized

console). For “cgfour” systems (3/60, 3/110, and 4/110 with the normal color monitor), you can use this to specify color-only, monochrome-only, or to use both by a kind of “virtual screen” arrangement. The recommended configuration for these systems is to use:

```
% openwin -dev /dev/cgfour0 -dev /dev/bwtwo
```

In this configuration, the color screen (the default screen, or screen zero) is "to the left of" the monochrome screen (screen one); that is, moving the mouse off the right side of the color screen moves you into the black-and-white screen (makes the monochrome screen appear on the monitor); moving off the left of the monochrome screen brings back the color screen. For more details, see the *openwin* and *xnews* manual pages in Part Three of this Guide.

You can now proceed to the section "Selecting and Using a Window" on page 35.

If, on Solaris 1 (SunOS 4.1.x), you get a message like

```
wiwindow: Base frame not passed parent window in environment"
Cannot create base frame. Process aborted.
```

then you have erroneously set your PATH search variable and are picking up the SunView version of some programs (see Appendix L, *Running SunView Applications on OpenWindows*). Ensure that the directory */usr/openwin/bin* is ahead of */usr/bin* and all will be well.

2.1.3 Starting on a 386 with SVR4

A good example of a flexible system is Dell Computer Corp's System V Release Four with X11R5. Merely by setting one line in a system file, you can customize the appearance of the overall system to Sun's OPEN LOOK, AT&T-OL, Motif, or the MIT look. Copy the file */usr/XR5/lib/xinit/xinitrc* to your home directory under the name *.xinitrc*, and edit it. Remove the comment character ("#") from before the line `WM=xvolwm`, and insert the comment character before the line `WM=mwm`, and you will get a Sun look, including the *cmdtool* terminal emulator, that is closest to most of the examples in this book. For an AT&T-OL look, including the OPEN LOOK version of *xterm*, un-comment the line `WM=olwsm`. With `WM=xvolwm`, the screen should look something like Figure 2-7.:

You can now proceed to the section "Selecting and Using a Window" on page 35.

2.1.4 Starting with an X Terminal

Most X terminals now use *xdm*, and so behave as described in the section "Logging In via the Special xdm Window". If you aren't running *xdm*, check your vendor's documentation to see if your X terminal supports the *xdm* protocol; if so, see that documentation or Volume Eight, *X Window System Administrator's Guide*, to configure it. If you have an older X terminal that isn't *xdm*-capable, you normally use the X terminal's built-in *telnet* or *rlogin* session capability to connect to the file server on which the clients reside. Then you can invoke your *.xinitrc* script either directly or through a simple shell script. Here is one example of a script, *xterminal*, that the author has used in the past. The key point is to note that we want to start all the normal clients, but we do **not** want to try to start a server, because the X terminal itself takes care of starting its own server when you turn it on.

```
#!/bin/sh
DISPLAY=${1-myxterm:0};export DISPLAY
OPENWINHOME=/usr/openwin; export OPENWINHOME
LD_LIBRARY_PATH=$OPENWINHOME/lib;export LD_LIBRARY_PATH
            PATH=$PATH:/usr/openwin/bin; export PATH
exec /bin/sh .xinitrc
```

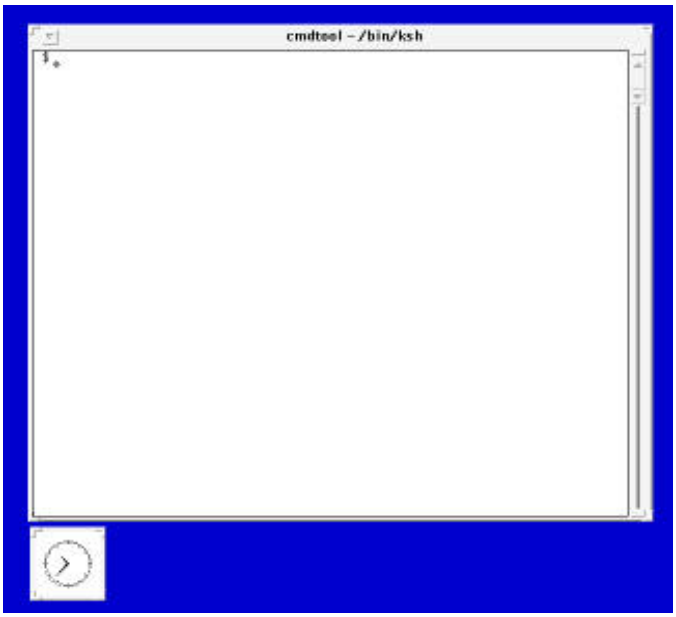


Figure 2-7. Initial screen with WM=xvolwm

In this code, `myxterm` is a default X terminal's TCP/IP hostname—the X terminal that I normally use. For more information on the DISPLAY variable, see Section 3.2.2, "Running a Client on Another Machine: Specifying the Display" on Page 61. The LD_LIBRARY_PATH and PATH settings are needed if you are using Sun OpenWindows clients (normally the *openwin* command sets this, but since we are invoking the clients without using that script, we have to set these explicitly; see also Section 3.2.5, "Complications: LD_LIBRARY_PATH). The last line starts up our *.xinitrc* script; the *exec* keyword is there for efficiency.

2.2 Selecting and Using a Window

Now that you have a terminal emulator and a file manager handy, you can begin using them. To use the terminal emulator, move the cursor into its titlebar and click the SELECT (left) pointer button anywhere except on the Menu Button at the left.†

The titlebar becomes highlighted to show that the window has been given the focus. We describe this "click-to-type" (or explicit focus) policy in Chapter One. (If *olwm* failed to start, or fails during operation, such that you have no window manager, you use the "pointer-focus" policy; simply move the pointer into a window and start typing.) If you start typing when no window has the focus, your keystrokes will be lost without warning.

† In fact, you can click anywhere in the window, but it's safest to click on the titlebar while you're learning the system.

Once you have the focus on a terminal emulator window, it should behave much like an old style ASCII terminal. The details vary depending on which terminal emulator you run and how you have configured it, of course.

From here on, we will be discussing the details of using the previously mentioned clients. The rest of this chapter discusses some specifics of using the OPEN LOOK GUI. The window manager, the file manager, and the terminal emulators will get our attention in the next few chapters.

There are two areas of any OPEN LOOK window that are used to provide feedback: the titlebar, and the footer. The titlebar is "grayed out" to indicate that the window will be busy for some time. And the footer is used to display messages. Watch for messages in the footer of any application using the OPEN LOOK GUI!

2.3 Creating Additional Windows

You can create additional windows whenever you need them. First we'll show you how to do so from within a shell window, then we'll see how you can create some windows from the system menus.

Move the input focus to the terminal emulator (*xterm* or *shelltool*) now. Since you have a UNIX shell running, you can run any program. One such program that you may need to run often is a second (or third...) copy of the same terminal emulator. Just type *shelltool* or *xterm*, an ampersand ("&"), and press the RETURN or ENTER key. This will create a new window, with a new copy of the UNIX shell in it (Bourne, Korn or C, whichever you normally use). It may take a few seconds, depending on how fast your system is. This window will have the default size for the program, typically 24 lines of 80 columns each for *xterm*, or 40 x 80 for *shelltool/cmdtool*.

By default, OPEN LOOK window managers will place new windows in a stack starting at the upper-left corner of the screen, and moving down and to the right a little each time you create a new window. We'll see later in this chapter how to specify the positioning of clients when you create them; this will apply to all clients, not just terminal windows. Note that a new window does not automatically get the input focus. With OPEN LOOK's default click-to-focus policy, you must click SELECT on a new terminal window before you can start using it. As well, anytime a window is partially hidden by another window in front of it, the SELECT operation that gives a window focus will also bring that window to the front, but only if you clicked on the window's titlebar. If instead you click inside the window, the system will give that window the input focus, but will *not* raise the window to the front. There are operations for explicitly moving a window to the back or front of other windows, as well as moving a window to a particular location, and others. We'll look at some of these shortly. But first, we need to learn some more details on using the pointer.

2.4 Using the Pointer

Most mice used with X11 have three buttons. OPEN LOOK defines their use as in the list that follows; this is similar to (but not identical to) the commonly accepted uses in most X

Window System applications. Note in particular that the *xterm* terminal emulator is *not* OPEN LOOK-compliant, so it uses some buttons differently. See Appendix A, *The xterm/olterm Terminal Emulator* for details.

- SELECT—the left button, used to select an object or text
- ADJUST—the center button, used to shorten or lengthen the selection
- MENU—the right button, used to access a Menu.

But what if you don't have a three-button mouse? The designers of OPEN LOOK anticipated this, and assigned these standard alternatives:

- On a two button mouse: Left is SELECT, shift-left is ADJUST, right is MENU.
- On a one button mouse: button is SELECT, shift-button is ADJUST, control-button is MENU.

2.4.1 Selecting Text

When using OPEN LOOK, you normally select something before you operate on it (this is called a "a selection-operation model"). For example, if you wanted to copy a sentence of text from one window to another, or to cut out a line of text from a window-based text editor, you would first SELECT the text, and then perform the cut or copy operation. The most common way of selecting a range of text is as follows:

1. Move the cursor to just before the start of the text
2. Click and hold the SELECT mouse button
3. Move or *wipe* the cursor across the text you wish to select (as you do, each character will be highlighted in inverse video)
4. Release the SELECT button

For example, let us assume that we have some text in a window, as shown in Figure 2-8.

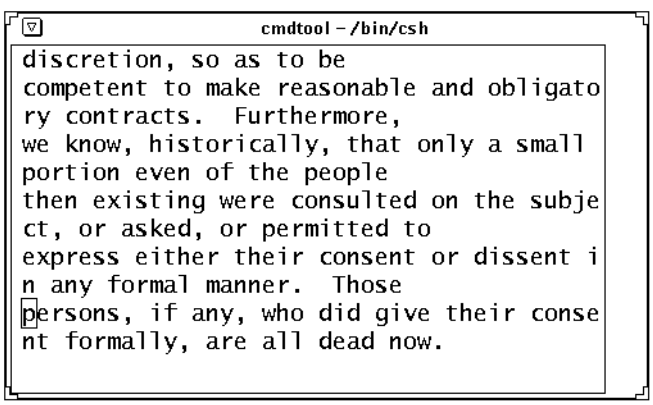


Figure 2-8. Text before selecting

Our goal is to select the words "only a small portion even of the people then existing were consulted." Move the pointer to just before the "o" of "only", then click and hold the SELECT button while you move the pointer to just after the "d" in the word "consulted". The screen should now look like Figure 2-9.

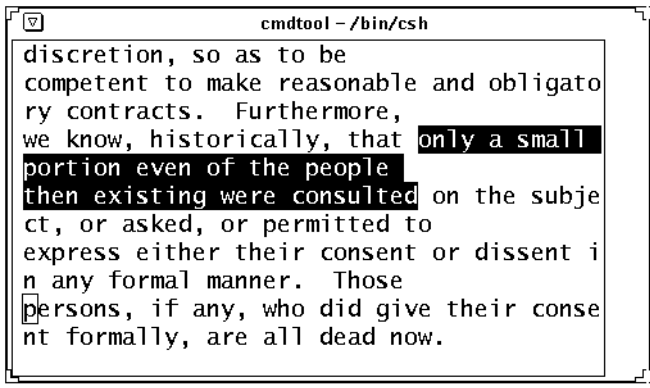


Figure 2-9. Text after selection

If you need to lengthen or shorten the selection *after* you've released the button, no problem. Just move the pointer to the new desired end point, and click ADJUST.†

An alternate method of selecting a range of text is to click and release SELECT at the start of the text you want selected, then move to the end and click ADJUST.

Then you can use the MENU button, or move to another window, to do the cut or paste operation. This will be described, with examples, in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, where we will also discuss other types of text selections. For now what matters is that text can be selected for use in copying, cutting, etc. Other objects can be selected, too: think of setting the input focus by SELECTing a particular window. You select the window and then do something to it (type into it, move it, etc.). In short, SELECTing text or graphical objects is a basic operation.

2.4.2 Menu Choices

Like most window systems, OPEN LOOK makes extensive use of on-screen "menus," or lists of related choices. OPEN LOOK programs generally don't have permanent menus, ones that are always present on the screen. Instead, they use pop-up menus. And in addition

† If you have previously used the MIT version of the *xterm* terminal emulator, you will know that *xterm* uses "button1" to start a selection, and "button3" to adjust it. But in OPEN LOOK, the ADJUST button is normally the middle button, "button2." There is no easy way to reconcile these incompatibilities, though individual users may overcome them with an X Toolkit mechanism called "translations" (see Appendix F, *X Toolkit Translation Table Syntax*). OpenWindows users may find it easier to use *cmdtool* instead of *xterm* while you are learning this material, since *cmdtool* conforms to OPEN LOOK instead of the *xterm* conventions. AT&T-OL, however, includes a version of *xterm* that *does* conform to the OPEN LOOK specification.

they have one of OPEN LOOK's significant contributions to window systems: "pinnable" menus, so that you the user can decide which menus are to stay up.

To use a menu, you must first display it with the MENU button (usually the right button). There are two ways to display OPEN LOOK menus, the traditional "pop-up sliding menu" style, and the "stay-up" style.

- Pushing the MENU button *and holding it down* "pops up" the menu—it stays on the screen as long as you hold the MENU button down while you decide what to do. You can move the pointer to any of the menu items and release it; this will cause the corresponding action to be invoked. As you slide the pointer up and down, the item under the pointer is highlighted, to indicate that it will take effect if you release the button. Figure 2-10 shows one such menu: this example is the normal "titlebar" menu for windows, or *Window Menu*, both as it first appears and after you move the pointer to the right to select `Close`.

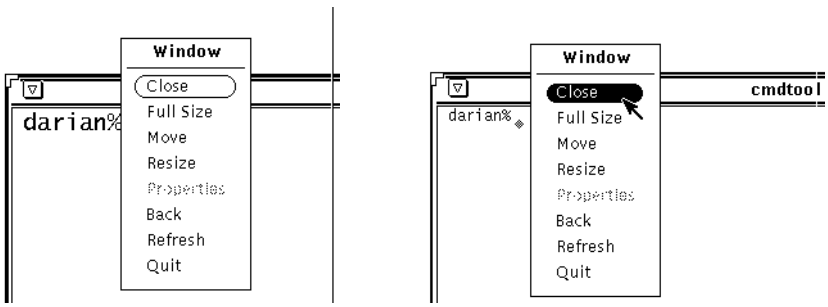


Figure 2-10. Window Menu

- On the other hand, if you push the MENU button *and quickly release it*, then the menu will appear in "stay-up" mode, that is, it will stay up just until you make one selection. You can click the SELECT pointer button on one of the menu choices to select that choice. If you want the menu to stay up for a while so you can invoke several of the actions, you can click the SELECT pointer button on the pin, which causes the menu to become pinned as described below†. Or, you can click any pointer button *anywhere else on the screen* (typically in the *Workspace* or background) and the menu will be dismissed.

Important menus, such as the "root menu" or *Workspace* menu, are *pinnable*, which means you can "pin them up." Some *windows* are pinnable too, as we'll see later. If an object is "pinnable", you can "pin" it, or make it stay up, just by moving the pointer over the picture of the pin and releasing the MENU button. You'll see the pin move before you release the

† Some implementations, such as Sun's *olwm*, allow you to click *any* pointer button to select a menu choice or pin the menu. Others do not, though, so you should consistently use the SELECT mouse button for these operations.

button, to show you that the pointer is in the right place. Figure 2-11 shows a pinnable menu before and after pinning.

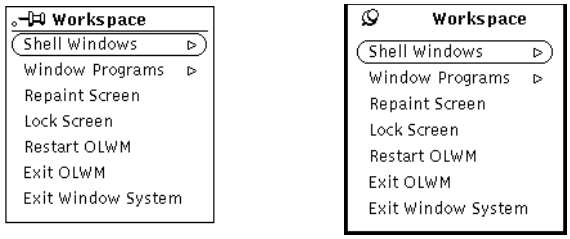
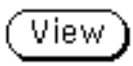


Figure 2-11. Pinnable menu before and after pinning.

The name comes by analogy with pinning up a note on your corkboard, or putting a Post-It note on your wall. Then the menu stays up with the pin in. If you later want to "dismiss" (get rid of) this menu, you can click the SELECT button on the pin, to pull the pin out and make the menu disappear. This discussion of pinning applies to any other item, such as a dialog box, that is pinnable.

2.4.3 Pushing Buttons and Menu Buttons

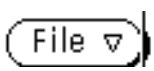
There are two types of buttons provided by the OPEN LOOK GUI, command buttons and menu buttons. Command buttons are the simplest; they look like this:



The function of a button will naturally be determined by the application. This particular button is from the *snapshot* program described in Chapter 9, *Graphics Clients*, and causes a "view" of the current image to appear. To invoke it, you simply move the pointer so that the point of the arrow is anywhere inside the oval, and click the SELECT pointer button More of OPEN LOOK's visual feedback comes into play when you push on any button: the button is grayed out until the operation it invokes is finished.

Note that some buttons have a "menu mark" consisting of three dots (" . . .") after the text. This indicates that when you select this button, a pop-up window (or "dialog box") will appear. For example, the "Print..." button is used to pop up a dialog that lets you specify which printer to use, number of copies, etc.

Many OPEN LOOK applications have buttons labelled *File*, *View*, *Edit*, and the like. These *menu buttons* have a pull-down arrow (▽) which means there is a pull-down menu attached to them, as shown below:†



† The *File* menu is normally used to open a new file, save your work in a file, etc. The *View* menu normally controls some aspect(s) of how the window is displayed. The *Edit* menu normally allows you to edit one or more aspects of the window's behavior, or edit multiple parts of the data. Watch for examples of these as we go, particularly in Chapter 7, *The OpenWindows DeskSet Clients*.

Clicking MENU on this button will pop the menu up, with all the flexibility discussed previously for menus. When you pull the menu down, you can that the item surrounded by an oval is the default. Buttons that do not have the pull-down indicator shown above are simpler—you just click on them with the SELECT pointer button when you want to activate them. The MENU button does not do anything on such buttons.

On the other hand, clicking the SELECT pointer button on it will select the default action, without having to view the menu. This is a convenient shortcut that you use after you're familiar with the button layout of a given application. But because you may not be sure, Menu Buttons also have a feature called "menu previewing." When you push and hold the SELECT button, they preview (display) the first few characters of the default item. If you then release the pointer button, the default action will be taken. If you move the pointer off that button, the default will not be taken. This is useful for applications that change their default (say, between *Load* and *Save* on their *File* menu button), or at any time you're confident but not certain about what the default action on a Menu Button will be.

2.4.4 Other Controls

In the following sections, we will discuss all of the common OPEN LOOK control items. Should you wish to experiment with these, run the "sampler" programs shipped with the OPEN LOOK GUI. These sample programs are aimed primarily at programmers who are developing applications using the widget set, and are similar to those offered by other widget sets, such as Motif. This section discusses the programs available with OpenWindows Version 3.0, but similar programs should be present with any implementation of the OPEN LOOK GUI. Look in *$OPENWINHOME/demo* to see what's available.

The first program, the OPEN LOOK Intrinsics Toolkit sampler, is in *$OPENWINHOME/demo/olitsampler* on OpenWindows. Figure 2-12 shows what it looked like on our system:

The second program is very similar to one provided with OSF/Motif. In both cases there is an attempt to provide a "periodic table" of the design elements, which parodies the original Periodic Table of the Elements used by chemists and physicists to classify all the basic chemical elements out of which all substances are built. In most such programs, all the pictures of movable controls, such as menus, scrollbars, and choices, are active, and can be used to experiment with the behavior of the controls. And, again for the benefit of the programmers in the crowd, the command button at the bottom of each small window pops up another window that reveals the source code needed to use that control. Figure 2-13 shows the OPEN LOOK Intrinsics Toolkit "periodic table":

Figure 2-13 is shown here in monochrome; to really appreciate how it looks, you should look at it in color, either on the page of Color Plates or by running it yourself on a system with a color monitor. Just give the command

```
/usr/openwin/demos/olittable &
```

and you should see a picture similar to the above. Try this now; we'll refer to some of the controls shown in the *olittable* program in the following discussion. You can ignore the two-letter names and numbers in the upper corners of each square; they are just there as a tongue-in-cheek reference to the original Periodic Table.

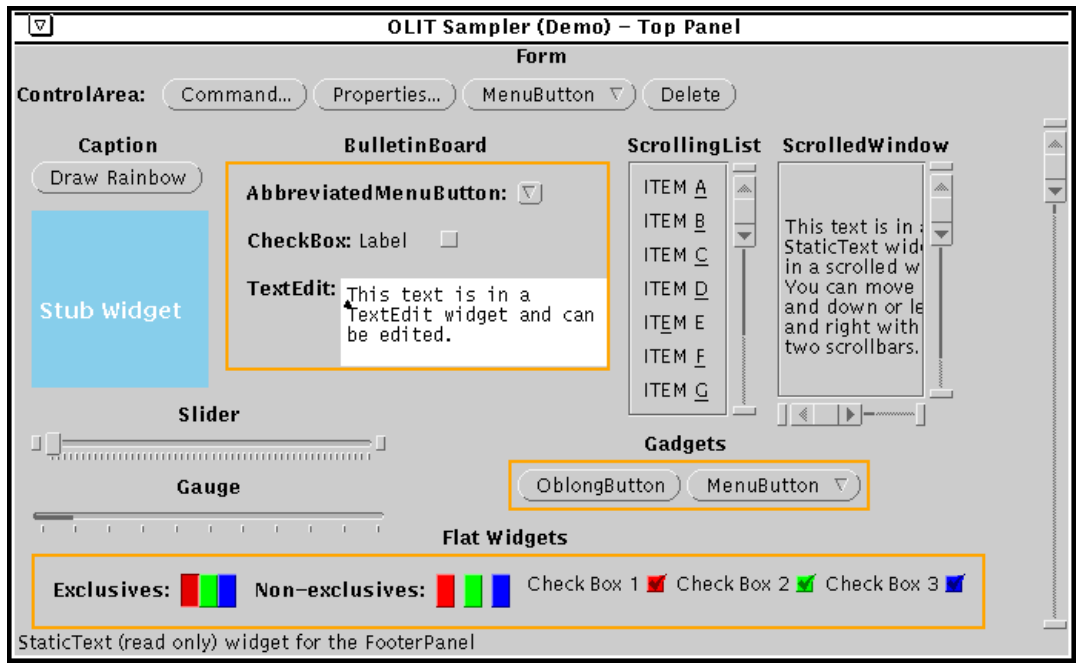


Figure 2-12. OLIT Sampler Program

2.4.5 Choice Items

Most window systems feature a variety of so-called "choice items", or controls that let you choose one or more options from a list. Here we look at the choice items provided by the OPEN LOOK GUI; exclusives and non-exclusives, checkboxes, and abbreviated choices.

2.4.5.1 Exclusives and Non-Exclusives

Exclusive and *Non-Exclusive* selections let you choose from a list of alternatives presented in small boxes. Each box is drawn with a darker outline (monochrome) or with a recessed image (three-dimensional look) when it is selected, and drawn with a thin border or drawn flat when not selected. The only significant difference between Exclusives and Non-Exclusives is that with the former, only one item can be selected, while with the latter, several or all of the selections can be active at once. Exclusive choices are drawn touching each other, to show that the buttons affect each other. For example, look at Figure 2-14, which shows more detail on the "Choices" section of *olittable*. The most common place to encounter these choice types is in "property sheets." A Property Sheet is a pop-up window that controls the "properties" or operation of a program. One example is the property sheet for *tapetool* shown in Chapter 7, *The OpenWindows DeskSet Clients*.

Each little rectangle, such as the one around *Pick Me* or *Bold*, is one "choice". The two sets of choices on the left are Non-Exclusives, while the two on the right are Exclusives. You can ignore the difference between the Flat and non-Flat versions of the choices; this is primarily for the benefit of application implementors.

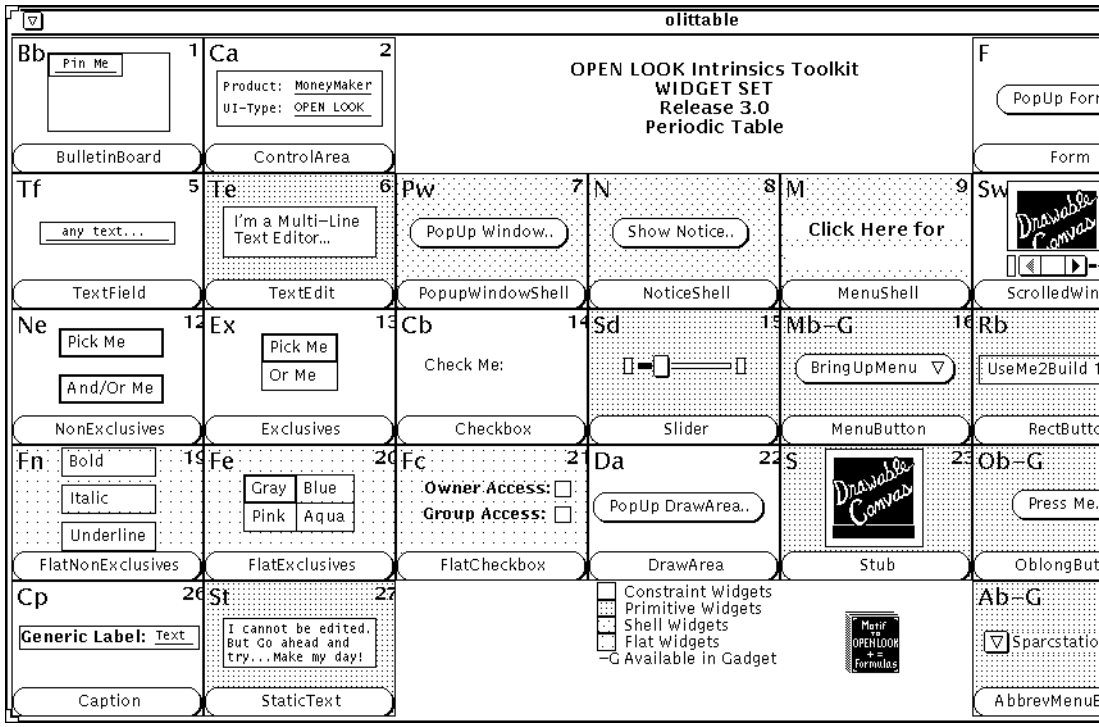


Figure 2-13. The OPEN LOOK "Periodic Table"

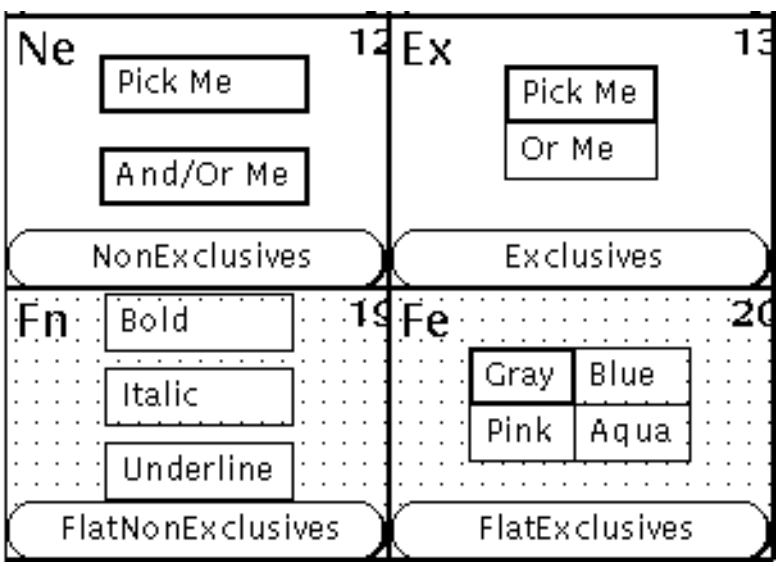


Figure 2-14. Choices section of olittable

Clicking SELECT on one of the Non-Exclusives will either select that item (if it is currently not selected), or deselected it (if it is currently selected). On the other hand, clicking SELECT on one of the Exclusives other than the one that is selected will select that choice and deselect the one that is currently selected.

For example, in a backup program, you might want to choose which tape format to use. It makes no sense to have both *ANSI Tar* and *CPIO* formats in effect at the same time, for example, since a tape can only be written in one format. So this decision would be an Exclusive choice. Figure 2-15 is a picture of such a (hypothetical) backup program:

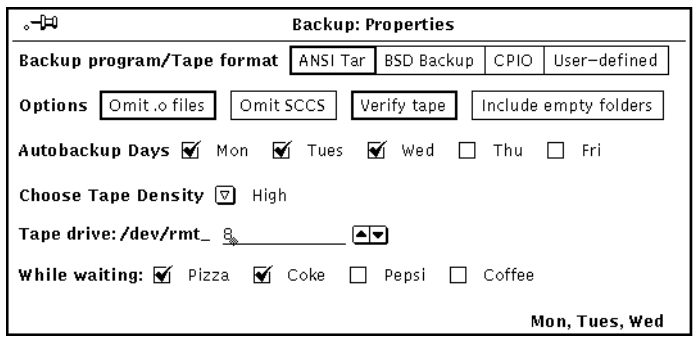


Figure 2-15. Choices in imaginary backup program

On the other hand, there are some options that are not mutually exclusive. Look at the Non-Exclusive settings in the Options row: it *does* make sense to turn on both "Omit .o files" (do not back up compiled "object" files) and "Omit SCCS" (do not back up SCCS archives), so either one, both, or any of the other options shown here can be turned on (or off again) by clicking SELECT in the box.

2.4.5.2 Checkbox

When you see a piece of paper with a list of items, some of which have little check marks beside them, you probably assume that those items with check marks have been done, or completed. The *Checkbox* control item is similar: use it to request that certain actions be done or that certain options be on or off. Figure 2-16 shows a close-up of the Checkbox item from *olittable*:

Figure 2-16. Checkbox from olittable

The Checkboxes are like little switches; if you SELECT them when they are on, they go off, and if you SELECT them when they are off, they go on. The third row of the program shown in Figure 2-15 allows you to turn backups on or off for each day of the week. The bottom row of this hypothetical backup program uses Checkboxes to let you order food and drink to consume while watching the tape drive spin.

2.4.5.3 Abbreviated Choice

The "Choose Tape Density" item is like an Exclusive choice, but it only displays the current setting. It otherwise functions like an abbreviated menu, discussed earlier in this chapter; if you click SELECT on the menu mark, the default is set, while if you click MENU, the menu pops up, as shown in Figure 2-17:

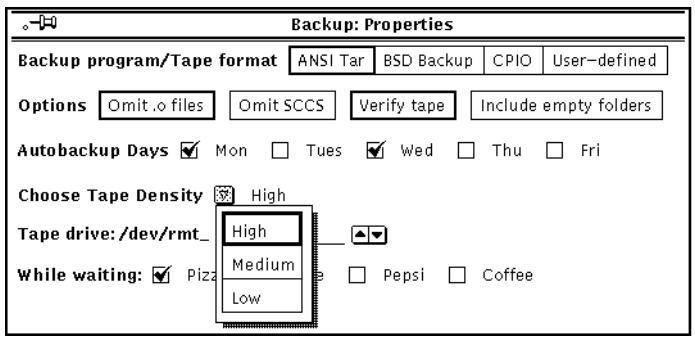


Figure 2-17. Abbreviated menu button pops up

Then you can click on any of the choices; the one you chose will be displayed beside the menu mark, and the menu will disappear.

2.4.6 Sliding items

There are several "sliding items" that let you scroll or adjust a value or a region. The *scrollbar*, *scrolling list* and *Gauge* controls are discussed together here.

2.4.6.1 Text Scrollbar

Scrollbars are one of the basic controls in any Graphical User Interface, because scrolling of a larger window over a small display area is such a common operation. You can scroll a command history list, a list of files, a list of colors to be used in displaying windows, or a list of paintbrushes in a drawing program. A consistent GUI such as OPEN LOOK makes it easy for the user by using the same scrolling notations throughout. Let's first look at the

scrollbar and how it is used for the common operation of scrolling text. First, we need to know the names of the parts of the scrollbar, as shown in Figure 2-18.

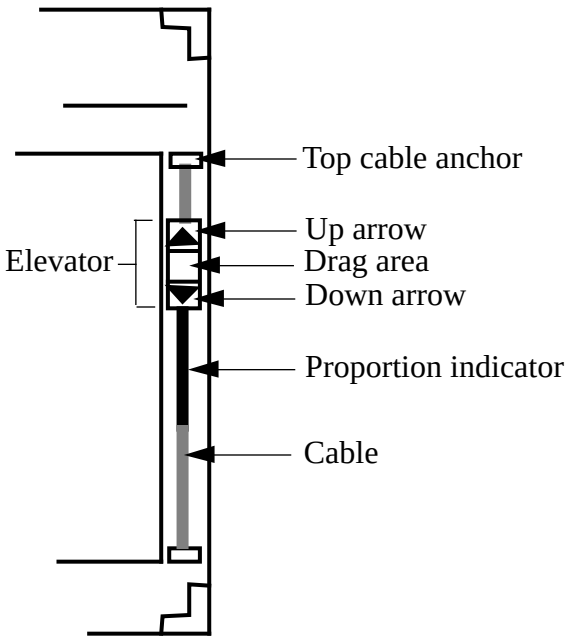


Figure 2-18. Parts of the OPEN LOOK Scrollbar

Here we present just the highlights of the scrollbar. A full explanation will be presented in Section 5.4, "Using the OPEN LOOK Scrollbar" on Page 108.

- To move the text up, click and hold SELECT in the centre box of the elevator, and drag it up or down. As you drag the elevator down, the text will move up, which means you are moving down in the file.
- To move the text down, drag the elevator upward.
- To move toward the top of the file, you can also click the little arrow on the top of the elevator; to move down, click the arrow on the bottom.
- To move up or down a "chunk" at a time (normally a page or half-page), click SELECT on the cable above or below the elevator.

There is much more to the OPEN LOOK scrollbar functionality, including a menu that lets you jump around and split the window into multiple views. We discuss the scrollbar in more detail in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, when we have some text to work with.

2.4.6.2 Scrolling List

The scrolling list lets you view several of a (potentially larger) number of choices, and select one or more of the items. Each item that you select (if the application allows multiple selections) is highlighted. You can visit parts of the list that aren't currently visible by appropriate use of the scrollbar, and select item(s) just by moving the pointer onto them

and clicking SELECT. So in a way, this type of item behaves like a cross between a text scrollbar and a vertical list of choices—you can scroll up or down, and you can select items from the list of choices. The *olitsampler* program shown earlier features a scrolling list, as does the simple window shown in Figure 2-18.

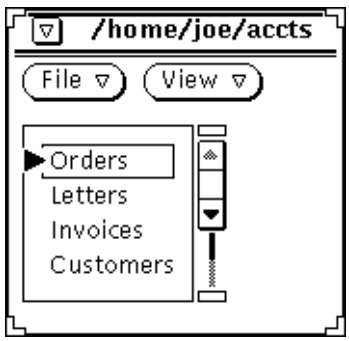


Figure 2-19. Scrolling list file browser

You can scroll the list up or down by using the scrollbar. In effect, the scrollbar here behaves pretty much the same whether it is scrolling around in a text file (scrollbar) or in a list (scrolling list). The Scrolling List scrollbar features a menu similar to the menu on a regular scrollbar.

At any point, you can select an item in the list just by clicking SELECT on its name. Like the Choices items discussed previously, Scrolling Lists can be either Exclusive or Non-Exclusive. In an Exclusive Scrolling List—commonly used for selecting one file or topic to open in an "Open..." pop-up window—only one item can be active at a time; selecting one deselects the currently active one. In a NonExclusive Scrolling List, each item can be selected or deselected individually.

A common shortcut is to double-click SELECT on an item to choose it, particularly where the scrolling list is being used to choose a file to open.

2.4.6.3 Slider

Like the scrollbar and scrolling list, a *slider* lets you move around. However, its purpose is not to actually move something, but to adjust a numerical value within a specified range. An example slider with its default value is shown in Figure 2-20:

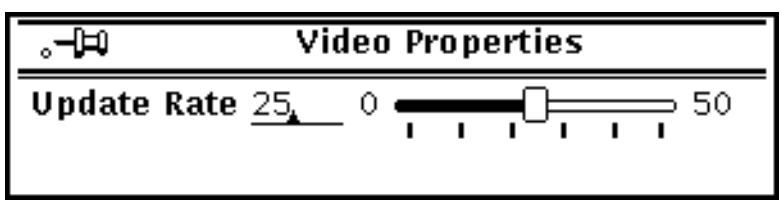


Figure 2-20. A slider

This slider has a range of from zero to fifty; it might represent frames per second on a video source, where zero means pause or "still frame," and positive numbers control the refresh rate. You adjust the value as with a scrollbar, either by dragging the central box or "grip,"

the analog of the scrollbar's elevator, or by clicking on either side of the grip. If you grab the grip by clicking and holding the SELECT pointer button on it, it gets highlighted, and you can drag it as much as you like until you let go (Figure 2-21).

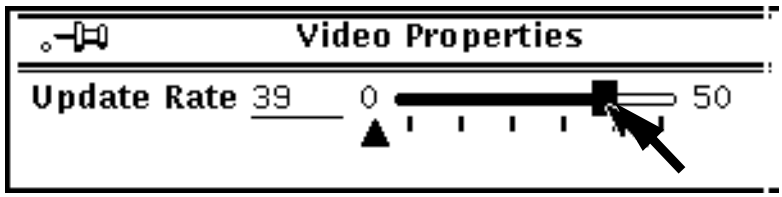


Figure 2-21. Slider being adjusted.

While most sliders are horizontal, some applications use vertical sliders.

The value shown numerically is in a text field, in which you can type an exact value (and press RETURN); this will set the slider to that value.

2.4.7 Gauge

The gauge item shown in Figure 2-22 is rather like a slider, except that it can only be set by the application. That is, it is like a "read-only" slider; you cannot move it with the mouse, but the program controlling the window sets its value to provide you with information.

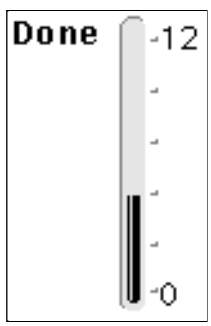


Figure 2-22. A gauge

2.4.8 Text Fields

There are several kinds of controls that let you manipulate text. A simple text field (see Figure 2-23) has a label and one line for insertion of text. The label is "Filesystem", and the current value is */dev/rsd0a.*

Text fields behave a little differently than other controls. To work with them, you must not only ensure that the window they are in has the input focus, but you must *also* ensure that the particular text field has the focus. You can do this by clicking the SELECT pointer button with the pointer in the text field; most applications will also allow you to move from one textfield to the next by use of the <TAB> key or some other accelerator. Once you have activated a text item, its *insertion point* ("caret" marker) will turn from gray to black to

Figure 2-23. A text field

indicate that it is active. Then, you can type text at the insertion point, or you can use any of several control keys:

- CTRL-U clears the text field
- CTRL-H, or Backspace, or DEL, erases one character at a time

A complete list appears in Section 5.7.1, “The Editing Keys” on Page 124.

Although a text field may only display a small number of characters, the application may be willing to accept more. Let’s say we have a version of UNIX that uses very long device names, one of which we are typing into a *Backup* program, and we use up all the character positions on the screen. This is shown in Figure 2-24.

Figure 2-24. Text field about to overflow

Nothing special happens yet, but as we type the next character, OPEN LOOK gives us more feedback, and lets us continue typing, by sliding the text to the left and giving us a *More arrow* at the left end (Figure 2-25).

Figure 2-25. Textfield with More arrow

If we wish to view text that is hidden behind a More arrow, we just click on the arrow, and characters will appear one at a time. If necessary, another More arrow will be created at the other edge of the text field. You can type as many characters as are needed in most applications; if you type more than the application is willing to accept, the text field will stop accepting input, and beep if you try to keep typing.

There are many other characters that can be used in text fields. As well, lines of a text field behave similarly to lines in an OPEN LOOK text window. There is much more detail on this topic in Section 5.6, "Editing Text in OPEN LOOK Applications" on Page 120.

2.4.9 Help Me

Although the OPEN LOOK GUI is remarkably consistent, there are quite a few different types of objects, and they can be assigned different meanings by the application programmer. So you need some way to find out what a particular control is for. Most window systems just tell you to "read the manual," or at best give you a "Help" function that is not aware of what you are doing (that is, they don't give you "context-sensitive" help). But the designers of OPEN LOOK knew that most users would want something more powerful: something based on window technology. So the OPEN LOOK GUI includes a facility for getting online, context-sensitive help. Put the pointer on any OPEN LOOK control, and press the HELP key. This key is labeled HELP on some keyboards (on a Sun-3 keyboard, it is labelled F1). If the developer of the application has provided a help file† and it has been installed correctly, you will see a window created to bring you the help data *about that particular OPEN LOOK control.* Figure 2-25 shows an example of a help screen for the default Workspace menu under OpenWindows. This is what you would see if you moved the pointer into the (pinned) Workspace menu and pressed the HELP key.

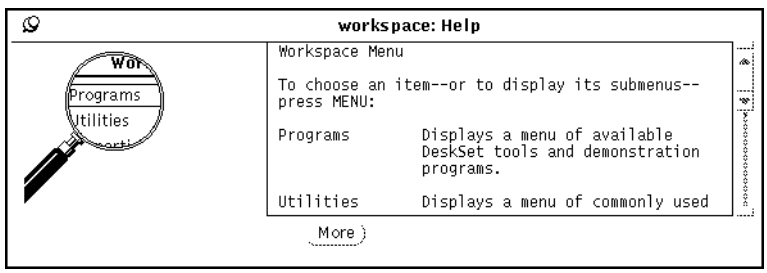


Figure 2-26. Help screen for a QUIT button

The Help screen includes a little magnifying glass to show what it is "zooming in" on; in this case, the Workspace menu. If you press HELP with the pointer over a command button, then only the help for that button will be displayed. If the help text is long, as in this example, there will be a text scrollbar, described a few pages ago. If there is no help file for a particular dialog item, or the help file for that application is not installed, you will see a small dialog box with the message "No help is available for . . ." and the name of the application and object for which the help is missing. For some commercial applications, you may either have to install the help file as part of the installation, or else set the environment variable HELPPATH to include the directory where the help files live. If the help function fails for one program, check that the help files were installed properly and that HELPPATH is set correctly.‡

† The help file for an XView application named *foot* has the filename *foot.info*.

You could also use the HELPPATH facility to provide your own alternate help files. If you wanted them to be more detailed or more terse than those provided, you could copy that program's help data into a private help library, put its name at the front of the HELPPATH list, and edit the copy of the help file. The facility is as flexible as the UNIX execution search path on which it is patterned.

One limit is that you cannot have part of one help file in one directory and part in another; that is, the system only opens one library help file per application name. Another is that some programs written with the OLIT toolkit may have the help data "hard coded," and you cannot provide your own help files for such applications. This is probably not such a good idea, but OLIT programs may include it, so beware.

2.5 Moving, Resizing, and Iconifying Windows

As with most window systems, you can use the titlebar to perform common window operations such as moving, resizing, and closing the window. Unlike many X11 window managers, however, OPEN LOOK lets you use the entire frame around the window to do these operations, so you don't have to have the titlebar visible. In the two-dimensional monochrome OPEN LOOK, the frame is very thin on the sides and bottom, but you can find it with a bit of practice. If you watch the screen cursor carefully, it "blinks" briefly as you move onto the frame around a window. In color OPEN LOOK, the frame is normally in a color that contrasts with the background and with other windows, so it's easier to see. You can set it to a different color using X resource variables, shown in *Section 3.4, "Customizing the X Environment: Specifying Resources" on Page 69*. Unlike some other window managers such as *twm*, you cannot change the border thickness under the OPEN LOOK GUI.

Once the pointer focus is in the titlebar or frame, you can use the MENU button to get the *Window menu*. The normal *Window* menu provided by the OPEN LOOK Window Manager has the following items: *Close*, *Full Size*, *Move*, *Resize*, *Properties*, *Back*, *Refresh*, and *Quit*. These will all be discussed in more detail in Chapter 6, *Using the OPEN LOOK Window Manager*; here is a brief look at the more commonly-used items:

- *Close* iconifies (or "minimizes") a window. The window disappears from your screen, and is replaced by a small window with the same name and (usually) a graphic representing the program. These icons appear in a row along one edge of your screen.
- *Back* moves the window behind any other windows that it overlaps. Useful in finding a window when you have several that overlap.
- *Refresh* redisplays the window, in case bits of it were overwritten or lost.
- *Quit* terminates the application; if the application has its own means of termination, such as logging out of a terminal emulator, that should be used instead of *Quit*. However, the presence of this item on the Window Menu is the reason why most OPEN LOOK applications do not provide a *Quit* button of their own.

‡ HELPPATH is a colon-separated list of directories to look in for help files. It defaults to */usr/lib/help* and */usr/openwin/lib/help* in AT&T-OL and Sun's OpenWindows respectively.

Figure 2-27 provides an example of closing a window using *Close*.

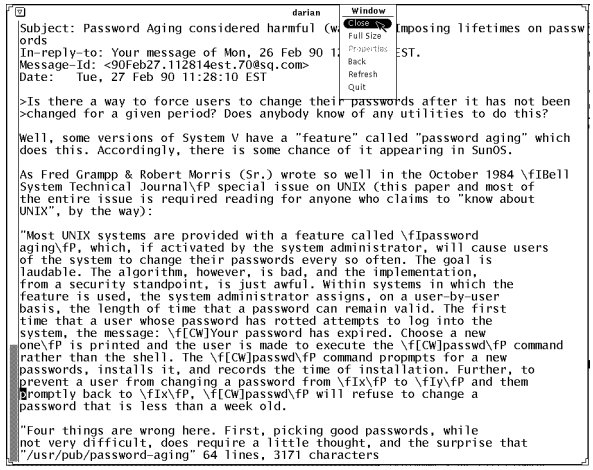


Figure 2-27. Closing a window from the window menu

See Section 6.4, "The Window Menu: Moving, reshaping, and iconifying Windows" on Page 144, for more details on all these operations.

2.6 Exiting an xterm or cmdtool Window

When you are finished using an *xterm* or *shelltool* window and want to get rid of it, you can remove in one of two ways. You can kill a terminal window by typing whatever command you usually use to log off your system. On UNIX this might be *exit* or CTRL-D. In Figure 2-28, we'll quit an *xterm* window by typing *exit*.

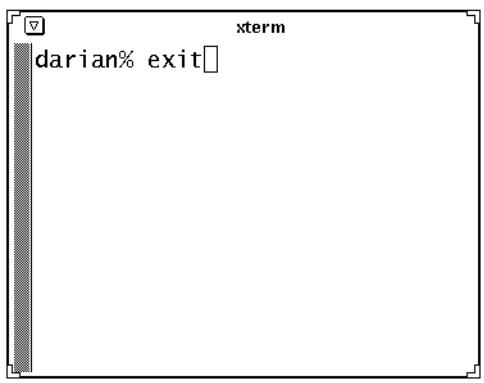


Figure 2-28. Exiting an xterm window.

A second, more drastic method of terminating the window is using the *Quit* item on the *Window* menu (see Figure 2-29). However, some programs that do not fully conform to the ICCCM (see Volume Zero, *X Protocol Reference Manual*), even including some versions of the MIT *xterm*, will give obnoxious messages when killed off this way.

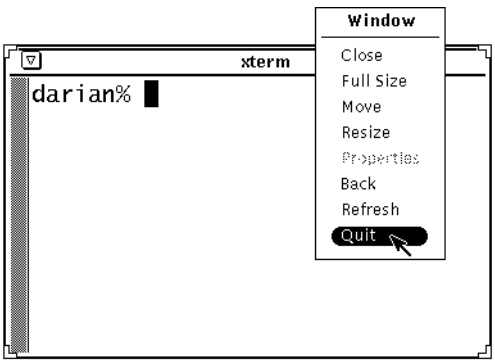


Figure 2-29. Quitting a window

Notice that in either case when we terminate the second terminal window, the first terminal window does not take over the input focus. When explicit focus is being used and a window is terminated, the input focus reverts to the Workspace, rather than to any particular window.

Be aware that on some systems (OpenWindows is not one of them), terminating the "login" terminal emulator window (the first terminal emulator to appear) kills the X server† and all associated clients. Be sure to finish any work you might be doing in all other *xterm* windows before terminating the *xterm* login window. For example, if you are in an editor such as *vi*, be sure to save your data before you terminate that window or the session. If you are on such a system, it may be wise to iconify the login window and use other *xterm* windows instead, so that you don't inadvertently terminate it. Remember: you iconify a window by placing the pointer on the pull-down symbol at the left end of the titlebar, and clicking the SELECT button.

If you are worried about typing CTRL-D accidentally, and you normally use the C shell (csh), you can enter:

```
% set ignoreeof
```

in the login window. Typing *exit* then becomes the only way you can terminate the window.

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

† If *xdm* is running X, the server will be reset but only after all client processes have been killed.

```
% unset autologout
```

The contributed software client *contool*, by Chuck Musciano of Harris Corporation, provides an alternative console window. It is normally iconified, to save "screen real estate," and can be set to flash or even pop open when important console messages appear. And it's an output-only window, so you can't accidentally kill it off by typing CTRL-D or exit in the window.†

For *xterm* only, an alternative way of logging off the system is to terminate an *xterm* window by selecting *Send HUP Signal*, *Send TERM Signal*, *Send KILL Signal* or (most reasonably), *Quit* from the *xterm Main Options* menu. (These menu options send different signals to the *xterm* process. Depending on what signals your operating system recognizes, some of the options may not work as intended. See Appendix A, *The xterm/olterm Terminal Emulator*, for more information.)

As we've seen, OPEN LOOK also provides the QUIT item in the *Window* menu to remove any client window, including an *xterm*.

2.7 Summary

Now you know the basics of managing windows under X11 and OPEN LOOK. In the next chapter we show you how to start many new windows, and how to customize the system so your desired windows start whenever you begin using the system.

† This program does require a certain feature in the operating system. To see if you can run it, check your vendor's documentation for the ioctl request TIOCCONS (4BSD) or SRIOCSREDIR (SVR4)— see the tty(4) or console(4) man page on your system—normally used by *cmdtool -C* or *xterm -C* to take over management of the console. If it's there, as it should be on any modern UNIX, you can use contool just by compiling and installing it. Otherwise, you or your support programmer would have to find out how *xterm -C* works on your system, if at all, and write similar code into *contool*.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 3

Opening Additional Windows

The previous chapter showed you how to start up OpenWindows or The X Window System and use the basic features of X11 and OPEN LOOK, using the OPEN LOOK Window Manager to control windows. In this chapter we show you how to:

- Start additional clients
- Start those clients in convenient places on the display.
- Customize a client process using command line options.
- Specify alternate default characteristics for a client program using “resource” variables.
- Arrange for your own particular set of windows to be created each time you start the window system.

3.1 Starting Additional Clients

Now that you know the basics of managing windows, you can start other X clients just as you can start another instance of the terminal emulator. The following sections describe how to open more client windows, place them on the display in convenient positions, and take advantage of X11s networking capabilities by running clients on other machines.

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

```
% oclock &
```

you can create a new window displaying a clock. The clock will appear in a the next spot in a stack coming down from the upper-left corner of the screen. You can then drag the *oclock* window to a more convenient location—say the upper-right corner, as in

Figure 3-1. (Remember, to move a window, place the pointer on the titlebar, press the SELECT button, drag the window outline to the desired location, and release the button.)

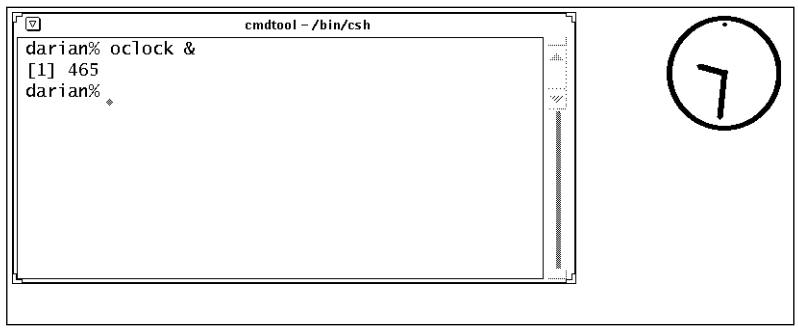


Figure 3-1. The oclock window

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

3.2 Command Line Options

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

The next few sections illustrate some typical uses of these important options. In explaining how to use them, we introduce some new, perhaps somewhat involved concepts (such as the way distances can be measured on your screen). Bear with us. Though these two options are not part of the OPEN LOOK specification, we feel that you need to master the `-geometry` and `-display` options in order to begin to take advantage of the flexibility of OPEN LOOK and X.

Most clients will accept a `-help` option, which will print the commonly-used command-line options that the given client accepts. XView clients often print only the "generic" options in response to `-help`. More detail on command line options appears in Chapter 11, *Command-line Options*.

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

3.2.1 Window Geometry: Specifying Size and Location

There is a command line option that is recognized by every imaginable X client to specify the size and placement of its base window on the screen. At first blush it looks a bit complicated. However, the OPEN LOOK Window Manager provides alternative methods of finding this information and saving it in a file. You can place your windows using the pointer, adjust their size using the pointer, then use the `Save Workspace` menu item. If you're willing to use that method, you can skip the remainder of this section until you need to know it.

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

```
-geometry geometry
```

The `-geometry` option can be (and sometimes is) abbreviated to as much as `-g`, unless the client accepts another option that begins with "g." Applications developed using the XView toolkit cannot use -g because of other options†. Specifying `-geometry` is always safe! Particularly in shell scripts, you should always use the long form, so that the script won't stop working when a new version of a program is installed; using the long forms also makes the scripts more readable.

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

```
widthxheight±xoff±yoff
```

Obviously, the first half of the string provides the ***width*** and ***height*** of the window. Many application windows are measured in pixels. However, application developers are encouraged to use units that are meaningful in terms of the application. For example, *xterm*'s dimensions are measured in columns and rows of font characters. More precisely, an *xterm* window is some number of characters wide by some number of lines high (80 characters wide by 24 lines high by default). *Cmdtool/shelltool*'s `geometry` dimensions are measured in pixels; use the alternate `-Ww` *x* and `-Wh` *x* to specify the size in columns and lines for *cmdtool* or any XView client whose base window is a text window.

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

Many users may not be accustomed to thinking in terms of pixels. What exactly is a pixel? A pixel is the smallest element of a display surface that can be addressed by a program. You can think of a pixel as one of the tiny dots that make up a graphic image, such as that displayed by a terminal, an X display, or even a home television set.

Since a pixel is so tiny, gauging sizes and distances in pixels will take some practice. For instance, what are the dimensions of your screen in pixels? Your workstation or X terminal

† If you *like* short forms, you can use **-WG** with XView clients; it takes the same geometry specification as does -geometry.

documentation may provide this information, or you may have to experiment by placing windows in order to figure out the approximate dimensions.

Keep in mind that monitors can vary substantially. The Sun 19-inch monitor usually has dimensions of 1152 x 900 pixels (these figures also comprise what is known as the screen's *resolution*). Since the horizontal and vertical dimensions of the viewing area are approximately 13.75 x 10.75 inches, each inch is equivalent to approximately 84 pixels (sometimes called dots) in both dimensions. By contrast, the 16-inch monitor on the Sony NEWS workstation has dimensions of 1280 x 1024 pixels. The actual viewing area is approximately 13 inches wide by 10 inches high. This translates to 98.5 dots per inch (dpi) horizontally and 102.4 dpi vertically, or roughly 100dpi.

Most X servers on PC-class machines require the use of a VGA monitor board. A VGA display has a minimum resolution of 640x480, and the 13-inch monitor we measured has a viewing area about 9-3/4 x 6-6/8 inches, giving a resolution of 65.5 x 72.5 dots per inch. Since most "VGA compatible" screen adapters also support other higher resolutions, *there is no single resolution standard for PC-class machines with VGA.* The same monitor in an 800x600 mode would have a resolution of 82 x 90.5 dots per inch. And at "Super VGA" 1024x768, the resolution would be a respectable 105 x 116 dots per inch.

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

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

If you don't specify the x and y coordinates at which to place a client window, the client may provide a default location; if the application doesn't provide a default, the *olwm* window manager will automatically position the window in a "stack" working down and to the right from the upper-left corner of the screen.

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

Table 3-1. Geometry Specification x and y Offset

Offset Variables	Description
`+xoff`	A positive x offset specifies the distance by which the left edge of the window is offset from the left side of the display.

Table 3-1. Geometry Specification x and y Offset

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

For example, the command line:

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

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

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

Table 3-2Geometry Specifications for Corners

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

If you want a window placed away from one or both edges of the screen, the guesswork starts. How many pixels away from the left side of the screen? How many pixels down from the top? You'll have to experiment with placing some clients on your screen to get a better idea of x and y offsets. And you'll have to rerun the experiments when you move to a different workstation with a different display board. This is why OPEN LOOK provides a stacked window placement policy, as described earlier.

It's actually a good idea to start some windows and move them around on your screen using the pointer. You can position some windows in different places by dragging them, and watching their sizes and positions. Under *olwm* you need only set the resources

```
OpenWindows.showMoveGeometry: True
OpenWindows.showResizeGeometry: True
```

Then *olwm* will display the position during a move, and the size during a resize. This information is displayed in a tiny window in the upper-left corner of the screen.

If for some reason you are not using *olwm*, you must figure out the window's x and y offsets using the *xwininfo* client. Chapter 8, *Other Standard Clients*, describes *xwininfo*, which gives you several statistics about a window including the coordinates of its upper-left corner.

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

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

This command creates a large *xterm* window in the lower-right corner of the screen, as illustrated in Figure 3-2.

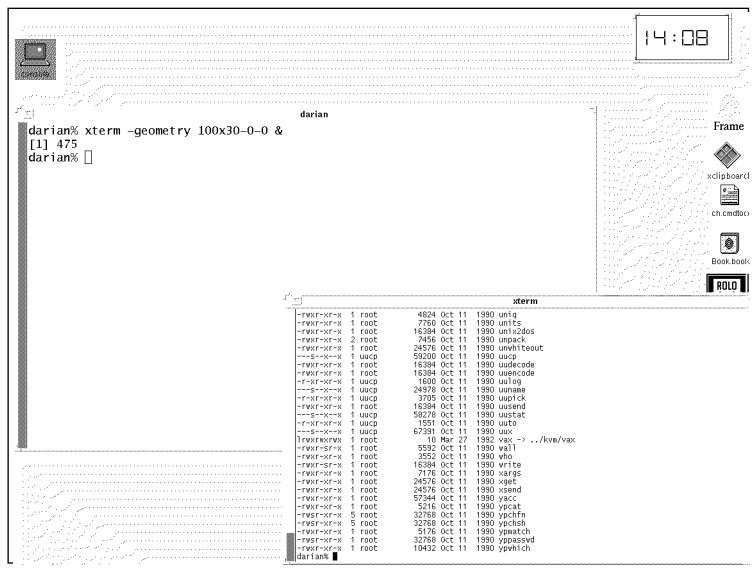


Figure 3-2. xterm window sized and positioned with the -geometry option

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

The geometry option is not the only means available to specify window size and location. Several clients, including *xterm*, allow you to set the size and location of a window (and often its icon or an alternate window) using resource variables (in an *.Xdefaults* or other defaults file). We'll introduce some of the basics of specifying resources later in this chapter. See Chapter 12, *Setting Resources*, for more detailed instructions on setting resources. See the appropriate client reference pages in Part Three of this guide for a complete list of resources used by any given client.

Clients built using the XView toolkit can also accept the command-line options `-Wp` x y to specify the window position in pixels, and `-WP` x y to specify its position when iconified.

✗ These XView-specific forms do not accept the + or - forms of position to indicate distance from the right margin or bottom. Furthermore, a bug in the XView library in OpenWindows 3.0 prevents use of the abbreviated "+xoff+yoff" forms of `-geometry` with negative values of *xoff* or *yoff*, that is, for setting the location relative to the bottom or the right margin. By experimentation we've determined which of the following forms work:

```
perfmeter -geometry -10-10fails (centre of screen!)
perfmeter -geometry 64x64-10-10 works (lower left corner)
perfmeter -geometry +10+10 works (upper left corner)
perfmeter -geometry +10-10 fails (10 from left, but centry of screen)
```

Accordingly, we recommend using the full-blown specification, even though it means hard-coding the size, with these clients, until the next release of the XView library.

✗ You should be aware that, as with all user preferences, you may not always get exactly what you ask for with any geometry request in any client. The X Window System requires that clients be designed to work with any window manager, not just *olwm*. If you are using some other window manager, it may have its own rules for window or icon size and placement. However, priority is always given to specific user requests, so you won't often be surprised.

3.2.2 Running a Client on Another Machine: Specifying the Display

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

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

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

However, by default a client creates its window on a display connected to the machine on which it is running. In order to run a client remotely and display its results locally, you must tell the client process where to display its window. For this purpose, X provides the `-display` command line option. Like `-geometry`, the `-display` option is recognized by almost all X clients. The display option tells the client on which server to display results (i.e., create its window). The option has the syntax:

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

In some clients the `-display` option can be abbreviated as `-d`, but it's safest to use the full form.†

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

In this context, "server" refers to a single physical display controlled by one X server program. (A display may be composed of multiple screens, but the screens share one keyboard and pointer.) Most workstations have only one keyboard and pointer and thus are classified as having only one display. Multi-user systems may have multiple independent displays, each running a server program. If only one display exists, as is the case with most workstations, it is numbered 0; if a machine has several displays, each is assigned a number (beginning with 0) when the X server for that display is started.

Similarly, if a single display is composed of multiple screens (sharing one keyboard and pointer), each screen is assigned a number (beginning with 0) when the server for that display is started. Multiple screen displays may be composed of two or more physical monitors. Alternatively, two screens might be defined as different ways of using the same physical monitor. One example is the Sun-3/60, Sun-3/110 and Sun-4/110 color workstations, described in Section 2.1.2, "Logging In at a Full Screen Prompt: Starting OpenWindows or X." PC/386 systems often support "virtual terminals;" on these you can sometimes run a separate X server on each of several virtual terminals. It is not clear whether there is a standard yet for referring to these as separate displays or separate screens; one expects that since there is but one keyboard, the server would treat it as one display with multiple screens.

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

Although much of the current X Window System documentation suggests that any of the parameters to the `-display` option can be omitted and will default to the local node, server and screen 0, respectively, we have not found this to be true. In our experience, only the ***host*** and ***screen*** parameters (and the period preceding ***screen***) can be omitted. The colon and ***server*** are necessary in all circumstances. For local hosts, the minimum form of the `-display` argument is thus*:0* and this form on some implementations may be more efficient than using *unix:0*, which some other documentation advises.

In summary, the display name can have the following forms:

† "X Toolkit" clients can usually use -d, but XView clients cannot because of the options '-disable_retained' and '-disable_xio_error_handler'. See the Reference Manual page for 'XView' in Part Three of this Guide.

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

On UNIX systems, the display name can also be stored in the DISPLAY environment vari-

Table 3-3. Display Syntax Forms

Display	Where window appears	How connected
(none)	Console of workstation you're using	Local socket (*/tmp/.X11-unix*)
:0 or :0.0	As above	As above
:0.1	Screen 1 of console	As above
unix:0.0	Console	As above
hostname:0.0	Console or screen of hostname	TCP/IP network (usually Ethernet)
hostname:0	As above	DECnet

able. Clients running on the local system access this variable to determine which display to connect to. The DISPLAY variable is set automatically, both by the OpenWindows start-up script and by the *xterm* terminal emulator. OpenWindows defaults it to `:0`, meaning the local host. In most circumstances, *xterm* sets the server and screen numbers to 0, and either omits a hostname (the local host is assumed) or sets the hostname to "unix," a generic name, which also defaults to the local host.

If you only run processes on the local machine, you don't have to deal with issues concerning the display setting. Clients running locally access the DISPLAY variable and open windows on a display connected to the local host. However, if you want to run a process on a remote machine and display the results locally, things become more complicated. A client running on a remote machine does not have access to the DISPLAY variable on the local machine. By default, a client running on a remote machine checks the DISPLAY setting on *that* machine, where DISPLAY may not even be set!

You can override the DISPLAY environment variable a client accesses by using the `-display` option when you run the command. Think of `-display` as a pointer to the physical display on which you want the window to appear. For example, say you're using a single display workstation and the display also has only one screen. The hostname of the workstation is *kansas*. In order to tell a client to connect to a display, you must identify it by its unique name on the network. (You cannot identify your display by the common shorthand—`unix:0.0`, `:0.0`, or some variation.) Let's assume that the complete display name for the workstation *kansas* is:

```
kansas:0.0
```

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

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

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

Keep in mind that for this process to succeed, you must have permission to run commands on the remote system (*oz*) *and* the remote system must have permission to "open" the local display (on *kansas*). If you don't have permission to run commands "over there," you will get back the message

```
Permission denied.
```

or words to that effect.

Normally you have permission to create windows from any machine on your local network onto your display, assuming that your local network shares home directories. The file *.Xauthority* in your home directory is created automatically by the *openwin* script and any other modern implementation of X11; this file grants permission to access your screen. It is not publicly readable so that other users cannot copy it. If your home directory is available on the remote machines from which you wish to launch clients (for example, a home directory mounted via NFS to a group of machines), no further action is necessary. But if you wish to grant permission to another account that has a *different* home directory but that you are "host equivalent" to, you can do so with the *rcp* command:

```
rcp .Xauthority otherhost:
```

If your account on *oz* has not been granted access to the server running on *kansas*, the window will not be opened, and you should get an error message similar to:

```
Xlib: connection to "oz:0.0" refused by server
Xlib: Client is not authorized to connect to server
Error: Can't Open display
```

On OpenWindows, this *additional* message will appear on the console of *oz*:

```
X11/NeWS Network security violation
 Rejected connection from: kansas (192.31.6.192)
For more information, see the xhost(1) and xauth(1) man pages
```

† The command to run the remote process might be different depending on the available networking software. Some versions of UNIX include a so-called "restricted shell" called *rsh*, so these systems have to invent a new name for the real *rsh*. If it's not there, or if you get the message `rsh: cannot open oz`, try *rcmd* or *remsh*. Ask your system administrator for the proper command.

If your command fails with this message, and you can't provide a *.Xauthority* file, then as a last restor try entering the command:

```
% xhost + oz
```

in a terminal emulator window running on your display. Then run the remote shell (*rsh*) again.

✗ Be aware that now *anybody* with an account on `oz` can create windows on your display; this is why the *.Xauthority* mechanism mentioned above is preferable. If problems persist, consult the *xhost* and *xauth* reference pages, or your system administrator, or refer to Volume Eight, *X Window System Administrator's Guide*.

This problem will also occur if you *su* to *root* and then try to create windows. Some vendors' software installation documentation tells you to use the "xhost +" form, but this is insecure. A better method is:

1. su to root
2. Copy *.Xauthority* from your home directory to */*.
3. Start the installation window.
4. Remove the file created in step 2!
5. Exit the root shell
6. Do the installation in the installation window.

The following command illustrates another common mistake:

```
% rsh oz cmdtool &
```

If you try to run a client using a remote shell and forget to direct the client to create its window on your own display, the window will not be displayed and you'll get an error message stating that the display cannot be opened. Worse, the terminal *may* appear on oz's console if an X server is running there; this gives whoever is logged in there access to your files on that machine!

In addition to specifying a local display, if permissions allow, you can also use the display option to open a window on someone else's display. You might want to display a window on another user's screen for instructional purposes. Multi-user systems can even be set up to allow teachers to display educational material simultaneously to several students, each using an X display of some sort.

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

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

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

3.2.3 Once You Run a Remote xterm using -display

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

✗ This automatic propagation is not done by the current version of *cmdtool* or *shelltool*

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

```
% echo $DISPLAY
```

The system should echo:

```
kansas:0.0
```

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

3.2.4 Logging In to a Remote System

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

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

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

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

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

3.2.5 Complications: LD_LIBRARY_PATH

Whenever you are starting a client program, you need to be sure that the shell variable LD_LIBRARY_PATH has been set. $LD_LIBRARY_PATH is the SunOS shared library search path, and is needed by most of the OpenWindows clients. This is normally set for you by the OpenWindows script, and normally propagated for you by the *rlogin* program.

Occasionally this fails, and programs refuse to start up.When this is the problem, you will get the message like

```
ld.so.1: programname: can't find file libxview.so.3
Killed.
```

Here are some circumstances in which you must set it manually

- You aren't using the *openwin* script to start your environment
- You have logged in remotely using *telnet* or any other network program that doesn't propagate your "environment variables"
- You are logging in from a non-Sun platform to a Sun platform to run some OpenWindows clients.

In these cases, the command

```
% setenv LD_LIBRARY_PATH /usr/openwin/lib
```

or

```
$ LD_LIBRARY_PATH=/usr/openwin/lib; export LD_LIBRARY_PATH
```

(typically from within a shell startup script) will solve the problem.

3.2.6 Monitoring the Load on a Remote System

There are two clients used to track the system load average on remote machines. These clients will be useful in determining which remote machine(s) on your network are least heavily loaded and therefore most suitable for logging into to get work done. The MIT client *xload*, described in Chapter 8, *Other Standard Clients*, tracks load average from one machine; the OpenWindows DeskSet client *perfmeter* described in Chapter 7, *The OpenWindows DeskSet Clients*, tracks several load-related statistics from a system.

3.2.7 Other Command Line Options

In Section 3.2.1, "Window Geometry: Specifying Size and Location" and Section 3.2.2, "Running a Client on Another Machine: Specifying the Display", we saw how to use the `-geometry` and `-display` options, which are accepted by most clients. Chapter 9, *Command Line Options*, describes some of the other options accepted by most of the standard clients. These options set window features such as:

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

Many clients also accept a large number of toolkit-specific options (listed in Chapter 11, *Command-line Options*) and application-specific options (listed on the reference page for each client in Part Three of this guide). Using a combination of standard and application-specific options, you can tailor a client to look and behave in ways that better suit your needs.

Command line options override the default characteristics of a client for the *single* client process. Traditional UNIX applications rely on command line options to allow users to

customize the way they work. X also offers many command line options, but these options have some limitations and liabilities.

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

X offers an alternative to customizing a single client process on the command line. You can specify default characteristics for a client using variables called *resources*, described in "Customizing the X Environment: Specifying Resources" on page 69.

Finally, remember that almost all clients will accept a `-help` option to give at least a summary of their command line options.

3.3 Putting it All Together

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

Say we're using a workstation with the hostname *jersey*. The workstation has a single display with one screen, so it will be called screen 0. Once X and the window manager are running, we might set up the display using the following commands.

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

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

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

```
% perfmeter -geometry 64x64-10-200 &
% perfmeter manhattan -geometry 64x64-10-20 \
          -display jersey:0 &
```

Run another *xterm* window on *jersey*.

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

Then iconify the login *xterm* window so that you don't inadvertently kill it (and shut down X in the bargain). Remember: to iconify a window place the pointer on the Window Mark on the window's frame and click the SELECT button. (See "Moving, Resizing and Iconifying Windows" earlier in this chapter.)

Run an *oclock*.

```
% oclock &
```

Figure 3-3 shows the *jersey* display, a fairly useful layout.

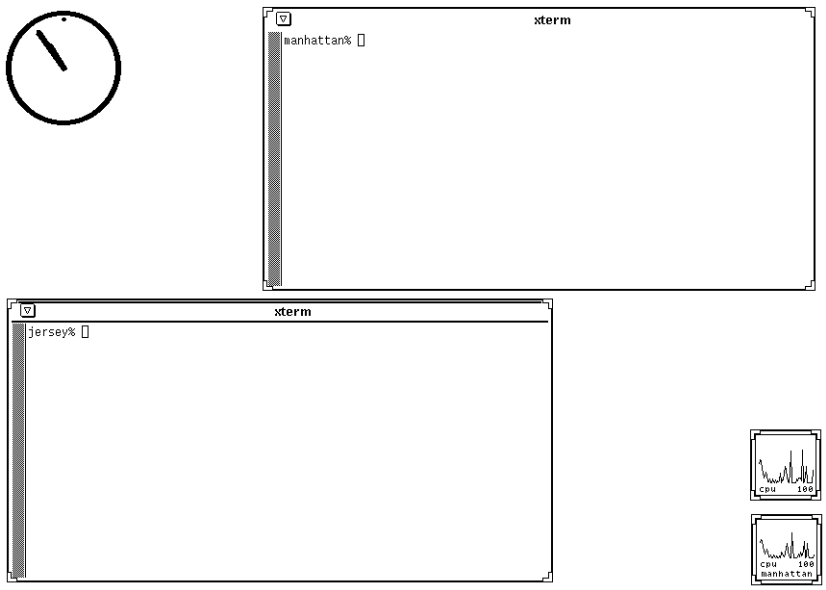


Figure 3-3. A working display

As we'll see in "Customizing your Session Start-up" on page 72, on most systems you can place the commands you run to set up your display in a special file that is invoked when you log in. Once this file (usually called *.xinitrc*, *.openwin-init*, or *.xsession*) is in place, your display will be set up to your specifications automatically each time you log in.

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

3.4 Customizing the X Environment: Specifying Resources

As we've seen, you can use command line options to specify certain characteristics for a single client process. In a sense, command line options allow you to customize one program.

X11 also provides a mechanism that allows you to specify characteristics that take effect *every* time you run a client. Almost every feature of a client program can be controlled using a variable called a *resource*. You can change the behavior or appearance of a program by changing the *value* associated with a resource variable.

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

A useful tool that the OPEN LOOK GUI provides is a *properties* program to edit your resource files for you, that is, to customize many aspects of your display. *Properties* is very easy to use, much easier than editing the resource files. The *properties* program is described in Section 14.1, "Properties Resource Editor" on Page 331. Accordingly, *users of the OPEN LOOK GUI on Sun OpenWindows do not need to learn about resource editing to configure the standard OPEN LOOK clients.* However you will eventually need to learn about resource editing for modifying extended features of the standard clients as well as most non-OPEN LOOK clients.

By default, the way a client looks and behaves is determined by the program code, and in some cases, by a system-wide file of *application defaults*. Several clients have application defaults files that determine certain client features.†

Application defaults files generally reside in the directory */usr/lib/X11/app-defaults* (on OpenWindows, in $OPENWINHOME*/lib/app-defaults*) and have the same name as the client application (except that the first letter is capitalized, and if the first letter is an X, the second letter is also capitalized). You can also have an application defaults file in your home directory, with the same format name. For example, the files */usr/openwin/lib/app-defaults/XEdit* and *$HOME/Bitmap* would both be valid names for application defaults files.

Need brief discussion of XFILESEARCHPATH—here, or in Resources chapter?

In describing the appearance and behavior of clients in this guide, we assume all of the standard application defaults files are present on your system and accessible by the client programs. If, by some chance, a client's application defaults file has been edited or removed from your system, the client may not look or behave exactly as we describe it. If a client application appears substantially different than depicted in this guide, you may be using a different version of the program or the application defaults may be different. Consult your system administrator if this causes problems.

Within an application defaults file, defaults are set using resources. The resources specified in a client's application defaults files are usually just a subset of a greater number of resources that can be set.

If the characteristics you set in your own resources file already have system-wide application defaults, your own settings take precedence. Keep in mind, however, that command

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

line options override both your own defaults and any system-wide defaults for the single client process.

To make your resource specifications available to all clients, X provides a program called *xrdb*, the X resource database manager. This program stores resources directly in the server where they are accessible to all clients, regardless of the machine the clients are running on. If you do not use *xrdb*, then each program reads the file *.Xdefaults* and (if it exists) *.Xdefaults-**host*** (where ***host*** is the name of the workstation whose display is being used), from your home directory. This is done by the lower-level "Xlib" routines, so it should work with all toolkits.

In some cases, a resource variable controls the same characteristic as a command line option. However, while the option specifies a characteristic for the single client process being invoked, a resource variable makes the characteristic the program default.

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

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

client**variable**: ***value

The following example shows five simple resource specifications for the *xclock* client. These particular resources specify attractive colors for *xclock.* See *Chapter 11, Command-line Options,* and the *xclock* reference page in Part Three for details.

Example 3-1. Resources to create a custom xclock

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

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

1. In your home directory, create a file containing the resources listed in Example 3-1. Name the file *.Xdefaults.* A resource file can actually have any name, but is often called *.Xresources* or *.Xdefaults*. *.Xdefaults* is preferable under OpenWindows since this file is maintained by the *Properties* program.

2. Load the resources into the server by entering the following command in a terminal window:

   % **xrdb -load .Xdefaults**

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

You should load resources using *xrdb* every time you log in. In the next section, we'll describe how to automate this process using a special start-up script, which also opens the client windows you want on your display.

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

Resource specifications can be much more complicated than our samples suggest. For applications written with an Intrinsics-based toolkit (such as the MIT Athena Widgets, AT&T's Open Look Intrinsics Toolkit, or the Open Software Foundation's Motif Toolkit), X allows you to specify different characteristics for individual components, or *widgets*, within the application. Typical widgets create graphical features such as menus, command buttons, dialog boxes, and scrollbars. Within most toolkit applications is a fairly complex widget hierarchy—widgets exist within widgets (e.g., a command button within a dialog box).

Resource naming syntax can parallel the widget hierarchy within an application. For instance, you might set different background colors for different command buttons and specify still another background color for the dialog box that encloses them. In such cases, the actual widget names are used within the resource specification. Chapter 12, *Setting Resources*, explains the resource naming syntax in greater detail and outlines the rules governing the precedence of resources.

3.5 Customizing your Session Start-up

By now you know that there has to be a file specifying the clients you want to run each time you login, otherwise you would have to start all your clients by hand. If you just start the X server itself, it will not start *any* clients for you, and you'll have to start the first terminal window from someplace else—not very convenient! The program *xinit* was designed to help you start all the clients you need. If you start *xinit* with no special preparation, it will try to start the default X server and at least one terminal emulator, but no window manager. Normally you would have a file called *.xinitrc* to tell *xinit* what clients to run. After *xinit* had been in use for a while, another program, *xdm*, was intended to solve the same problem in a different way. The *xdm* program is more sophisticated, but more complicated. Normal users need only be aware of one file that it uses, though: the *.xsession* file. Like the *.xinitrc* file, this file is a list of the clients that you want run each time you login via *xdm*.

Let's look at these files, and some others that are important for customizing your window system environment. Bear in mind that we are just going to scratch the surface here. For the full scoop on *xinit*, *xdm* and all their customization, you should refer to Volume Eight, *X Window System Administrator's Guide*.

The *.xinitrc* and *.xsession* files contain the clients you want run. But this script—whichever of the two you are using—should also:

- Set the DISPLAY environment variable
- Load your X resources file with *xrdb*.
- Start the window manager *olwm*.
- Start any terminal emulator windows, desk accessories, DeskSet tools, or X applications (text processors, spreadsheets, . . .) that you want to run.

Most of your clients will not interact directly with *xinit* or *xdm*, so they should be run "in the background", that is, with an ampersand ("&") at the end of each command. If you forget this, the shell (command interpreter) that is processing your start-up file will wait until the non-backgrounded client waits, before starting the remaining clients. However, if you background *everything* in the script, the shell will get to the end of the file and, with nothing more to do, terminate itself or "exit". This will cause the X server to terminate. Because of this, you may need to run a single console terminal window that is not put into the background. Alternately, you can use the *wait* command as the last line of your script.

Example 3-2 contains a listing of a simple example *.xinitrc* file:

Example 3-2. Simple .xinitrc file

```
#!/bin/sh

DISPLAY=myworkstation:0; export DISPLAY

# xterm -geometry 80x6+0+75 -T Console -C &
contool -Ws 550 160 -Wp 0 0 -WP 5 80 -Wi &

xrdb -load < $HOME/.Xdefaults
xset m 8 fp+ /usr/local/lib/fonts # bc

olwm &

clock -Wn -Wp 930 90 -digital -24 &

# Some terminal windows for working in.
cmdtool -Ww 80 -Wh 24 -Wp 000 200 -Wl "$hostname" -scale extra_large &
#xterm -geometry 80x48+000+176 -T "$hostname"&
#xterm -geometry 80x48+580+180 -T sq -e rlogin sq &

maker -geometry -0+900 & # Run FrameMaker if licensed

wait # forever, or until killall or shutdown.
```

We've used the Bourne shell (*/bin/sh*) as it is the shell most commonly used for writing scripts. You can use another shell such as *csh* if you are more comfortable writing scripts in it.

It is common to find commented-out entries (lines that begin with a "#") when looking at another person's *.xinitrc* file, so we've left some in. Such lines are ignored by the system.

The main thing is that all the possible command-line options discussed previously (as well as those in Chapter 11, *Command-line Options*) can be used, so this mechanism represents a powerful and flexible facility. If you may be using both *xdm* and either *xinit* or the *openwin* start-up script, you should consider keeping your start-up clients in a third file that is included in either your *.xsession* or your *.xinitrc* file. This is in fact what the default *openwin* start-up files do.

As well, it is possible without using *xdm* to have your X11 session started automatically when you login, simply by invoking it from your *.login* or *.profile* startup file. Example 3-3 is a fragment of the default *.login* file used in the current releases of SunOS. This code will normally start the *openwin* script, but gives you a chance to interrupt it first (during the `sleep 5` sequence), and logs you out afterwards, again with a chance to interrupt. This has proven to be quite convenient.

Example 3-3. SunOS .login code to start OpenWindows at each login

```
switch( $mychoice ) # "mychoice" is either "openwin", "sunview" or null
case openwin:
    echo -n "Starting OpenWindows (type Control-C to interrupt)"
    sleep 5
    $OPENWINHOME/bin/openwin    # run the window system
    clear   # get rid of annoying cursor rectangle
    echo -n "Automatically logging out (type Control-C to interrupt)"
    sleep 5
    logout  # logout after leaving windows system
    breaksw
    #
# other cases here
endsw
```

You may wish to incorporate code similar to this in your shell-specific startup script.

3.5.1 OpenWindows Specifics

OpenWindows uses a variation on the above files. If you already have a *.xinitrc* file, you can continue to use it, since OpenWindows does use *xinit* to start its server. But you should add a few entries gleaned from reading the OpenWindows-provided files mentioned below, otherwise certain aspects of OpenWindows's interface may not behave correctly.

If you do not have a *.xinitrc* file in your home directory, then OpenWindows will create one for you. The file it creates is fairly short; it calls an extra file named *.openwin-init* to start your clients. The file it creates is copied from */usr/openwin/lib/Xinitrc*, and does the following steps:

- Runs *xrdb* on *$HOME/.Xdefaults* or *$OPENWINHOME/lib/Xdefaults*
- Includes *$OPENWINHOME/lib/openwin-sys*, another script
- If you haven't requested otherwise, enables SunView binary compatibility mode.
- Starts the OPEN LOOK Window Manager and waits until it has finished initializing

- Runs the client start-up script, either *$HOME/.openwin-init* or *$OPENWIN-HOME/lib/openwin-init.*
- Waits for *olwm* (the "key client" or "session gate client") to exit.

While this may seem like substantial added complexity, it has two advantages:

- It works correctly even if the user has no start-up files to begin with, and
- It is easy to update the list of clients and their arguments, since they're in a separate file.

This technique of having the clients separate from the setup information is a generally useful one. It can also be used with any of OpenWindows, *xinit*, or *xdm*.

In fact, this is the reason that the OPEN LOOK Window Manager has a built-in operator called "SAVE_WORKSPACE", called from its default `Utilities` menu, that saves the list of all known X clients, their positions, and their arguments, into your *.openwin-init* file. OpenWindows users can therefore forget all about ***.****xinitrc* files, and simply use the "SAVE_WORKSPACE" mechanism to keep their list of clients. Or, they can use all the flexibility that the ***.****xinitrc* file provides, by using their own file in place of the one that OpenWindows provides for you.

✗ The "SAVE WORKSPACE" option produces a file that may not contain certain non-OPEN LOOK, non-X11 clients, and that may contain additional detail beyond what you would normally want to specify.

You should also know about the OPEN LOOK Window Manager menu file, *.openwin-menu*, used to control the workspace or root menu from *olwm*. This is described in detail in Chapter 13, *Customizing olwm*

3.6 Where to Go From Here

We've introduced some basic operations you can perform using OPEN LOOK and the *olwm* window manager. There are many useful client programs supplied with OPEN LOOK and the X Window System. Details of how to use two of the most important clients, file manager and the terminal emulator, are provided in Chapter 4, *Using the OPEN LOOK File Manager*, and Chapter 5, *The cmdtool/shelltool Terminal Emulator* (the MIT terminal emulator is described in Appendix A, *The xterm/olterm Terminal Emulator*). For instructions on performing additional window manager operations, read Chapter 6, *Using the OPEN LOOK Window Manager*. An overview and tutorial for OpenWindows and other standard clients is provided in Chapter 7, *The OpenWindows DeskSet Clients*, and Chapter 8, *Other Standard Clients.* Chapter 9, *Graphics Clients*, describes several graphics utilities available with X. Part Two of this guide provides information on customizing your X environment. All clients are described in detail in a reference page format (basically the UNIX *man* page format) in Part Three of this guide.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 4

The OPEN LOOK File Manager

The OPEN LOOK GUI is unique among X11 Graphical User Interfaces in specifying the detailed behavior of important client programs such as the File Manager. With MIT X11, there is no comparable program. With Motif, there may or may not be a version of one of several competing file managers or "desktop managers" like *X.Desktop* or *Looking Glass*. But with any implementation of a complete OPEN LOOK environment, there should be a File Manager program, and it should behave as described in this chapter.

The File Manager is an important tool for users accustomed to icon-based systems such as the Macintosh and Microsoft Windows, as well as to those unaccustomed to command-line oriented systems with hierarchical directories, such as UNIX. The File Manager is also a good illustration of how to use the standard OPEN LOOK "drag and drop" facilities.

Sun documents the File Manager as part of the DeskSet suite of tools, and by this logic it should be described as part of Chapter 7, *The OpenWindows DeskSet Clients*. However, we feel that a program of this sort is of sufficient import to justify its own chapter, so here it is.

4.1 What is a File Manager?

The File Manager is a program that makes it easy to manage your files. When you start the File Manager, files and directories are represented pictorially by small images called *icons*. You can work on these files and directories by manipulating the icons that represent them. You can think of the File Manager as an iconic directory browser, or as a visually-oriented command interpreter (or shell), or as a tool for making your OPEN LOOK system look more user-friendly. You can even think of it as a tool for turning a $5,000 workstation into a $1,000 MacIntosh™ (but this will not impress your financial advisors.)

In the Sun environment, it is also an important part of The X Window System environment in that it enhances some of the other DescSet tools, which therefore don't need, and don't have, very fancy file/directory browsing capability; you can drag mail messages to direc-

tories in the File Manager, or drag files from it to PrintTool, and so on. Also, under Solaris 2, File Manager interacts quite well the the Volume Manager, so that you need only insert a floppy or CD-ROM (and if it's a floppy run the *volcheck* command or the *File->Check For Floppy* menu item, and you will see a new directory view of the floppy. You can also format or relabel floppies, unmount and eject them, etc.

The File Manager is all these, and more. Since File Manager is a visual program, let's start with a picture. Figure 4-1 shows a screen with a File Manager running, and a row of icons.

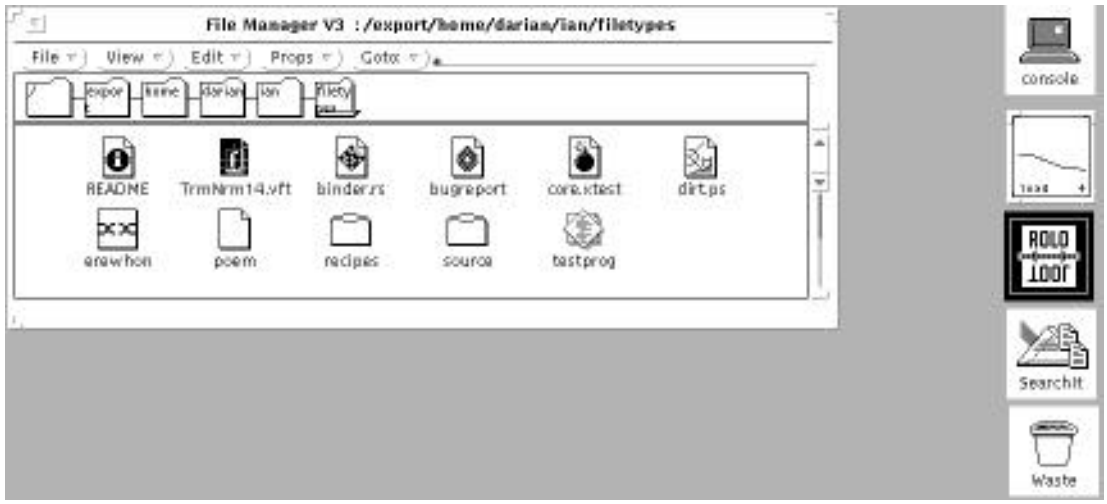


Figure 4-1. The File Manager

This gives an overview of some features the File Manager offers: a menu bar of pull-down menus to control its operation, a map from the "root" of the filesystem ("/") to the current directory, and a display pane showing the name and an iconic picture for each file object in that directory.

The icons *inside* the File Manager window represent *files*, some of which may be executable programs waiting to be run. The icons *outside* the File Manager, that is, on the Root Window or Workspace, represent *running programs*. This distinction is important. Unlike the MacIntosh Operating System, neither UNIX nor the OPEN LOOK GUI have the notion of a single current application; each known type of file will automatically be opened by the correct application when you double click on it.

In this example directory, Table 4-1 lists what is in each of those files.:

Table 4-1. What's in a Name?

Filename	Data type
README	English text
TrmNrm14.vft:	font definition
binder.rs:	rasterfile, 64x64x1 standard format image

Table 4-1. What's in a Name?

bugreport	Frame Maker document
core.xtest	core file from xtest
dirt.ps:	PostScript document
erewhon:	broken symbolic link to badsym
poem:	english text
recipes:	directory
source:	directory
testprog:	executable shell script

In short, the File Manager shows a pictorial representation of all the files. Note that all file icons have the same overall shape — that of a page with the corner folded down — but those whose contents are identifiable have a more detailed picture inside. One exception is executable programs which are shown with a special icon (see above *testprog* in Figure 4-1).† And all directories are shown by pictures of a file folder. File Manager uses this analogy throughout; directories are called "folders" in the File Manager screens and documentation.

But there is much more to the File Manager than is shown here, including the ability to drag-and-drop a file to another application. File Manager provides a visual metaphor for such tasks as navigating the directory tree, moving or copying files from one directory to another, deleting files, and many other operations.

If you are an old-time UNIX diehard, you may dislike such visual imagery, and prefer the command-line mode of using systems. You'll probably find some familiarity with the File Manager useful for drag and drop occasionally, but you may just want to skim this chapter.

4.2 Common Operations

If you see an icon representing a file and you want to do "the obvious" operation on it, then all you have to do is double-click SELECT on its icon, and the appropriate action will happen. For example, the file *poem* in this directory can be edited just by moving the pointer onto the picture of a file labelled "poem," and double-clicking the SELECT button. An editor will be started with *poem* as the input file. Let's look at some of these mouse operations in more detail.

† In OpenWindows 2.0 they were shown as a tiny OPEN LOOK window since they will run in a window. (IBM-PC and Macintosh users may choose to think that executable programs have been drawn as a picture of a floppy disk).

4.2.1 Running programs

Double-clicking SELECT on just about any file's icon will cause a program to be run. If you click on an icon that represents an *executable* object — either a shell script or a compiled program — the File Manager will try to run it for you. On the other hand, clicking on a file whose type is *bound* to some application will cause that application to be run. *Binding* a file to an application means that all files of a particular type will be run through that application when selected from within the File Manager. This *binding* between files and actions is just a name for what happens when you click on a file's icon. For example, files of type "worksheet" might be *bound* to the Lotus 1-2-3 application. This means that when you double-click on the icon for a "worksheet" file, the File Manager will automatically invoke Lotus 1-2-3 on that file.

Many standard types are bound for you automatically, including many of the standard commercial applications available on the Sun platform. We'll see some of these in the following paragraphs, and how to change the bindings (or even add your own) in *Section 4.4, "Customizing the File Manager,"* later in this chapter.

First, let's look at running an application that is in our directory. The file *testprog* shows up as an executable program. Let's double-click on it. It gets marked as selected or busy (the icon is shown in reverse video), then the footer of the File Manager window displays this message:

```
Opening testprog; errors may display on console or cmdtool.
```

If you requested a window-based application, you will see that program's main window appear. If your application is not window-based, but is a simple command-line program, you won't necessarily see any further indication that your program is running: File Manager won't give you any; beyond the notification that it has started your program. If the program writes to the UNIX *standard output*, its output will normally appear in your Console window. In either case, File Manager is available for other work as soon as it has *started* your application; you don't have to wait for the application to finish before using other File Manager functions.

What about starting up a program by clicking on a file whose type is bound to an application? In our sample directory, the first two files, *TrmNrm14.vft*, and *binder.rs*, show icons that are associated with graphical utilities; the former is a binary font file in a form that can be edited using Sun's *fontedit* program (see Chapter 9, *Graphics Clients*). The latter is a raster file that can be displayed with the *snapshot* program, also described in Chapter 8. The file *dirt.ps* is a PostScript file, and clicking on it (or any file that is recognized as containing PostScript) will start the OpenWindows PostScript previewer, *pageview*, to view this file (see Chapter 9, *Graphics Clients*). The file "poem" is a text file, so it is considered to be bound to an X-based text editor, normally *textedit*, (see Chapter 5, *The cmdtool/shelltool Terminal Emulator*). Double clicking on any of these icons will try to launch the corresponding application for you.

4.2.2 Change Directory

Eventually you will get tired of looking at the files in just one directory. If a directory icon (picture of a file folder) is shown in your File Manager window (as are *recipes* and *source*

in our sample directory), you can change directory to it just by double-clicking SELECT on it. A short form for this is to click SELECT once and hit the RETURN key. The File Manager will do a change directory operation internally, and will then read the contents of the new directory and display it, using the same iconic representation as before. Note that this change of directory does *not* affect any other shells or programs you may have running in other windows; on UNIX there is no way for one process to change another's working directory. It does, however, set the working directory of any new programs or applications that you start from within this instance of the File Manager.

Let's say you double click on the folder (directory) *recipes*. The icon for that directory will show busy for a second, then File Manager will display the new directory, and your new screen might look like Figure 4-2.

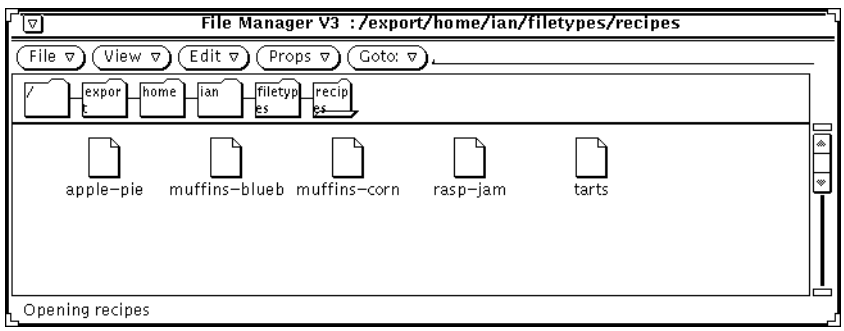


Figure 4-2. File Manager in recipes directory.

Then you might wish to edit the "tarts" recipe. Just double-click on the picture of the file labelled *tarts.* Your File Manager titlebar will show "busy" for a moment, as in Figure 4-3.

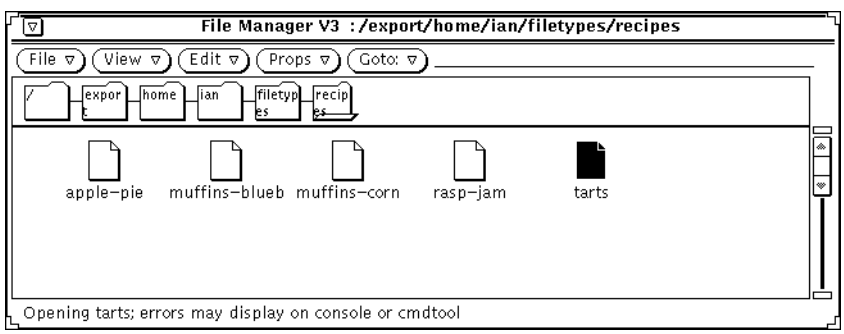


Figure 4-3. File Manager starting editor on "tarts"

Figure 4-4 shows the edit window that will appear in a few seconds.

On OpenWindows, the editor used by default is *textedit*, described in Chapter 5, *The cmdtool/shelltool Terminal Emulator* (To change this default, see "*Customizing the File*

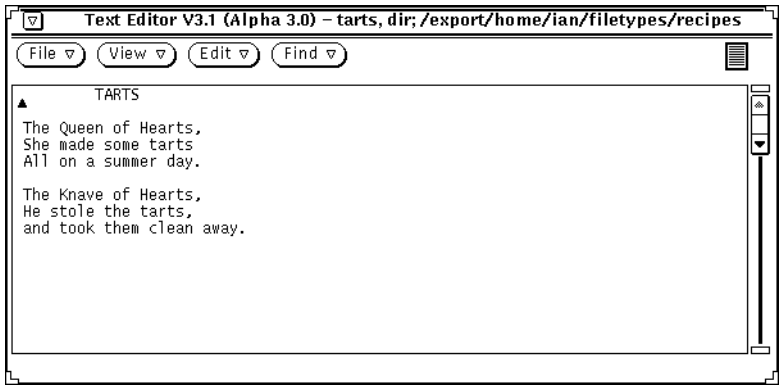


Figure 4-4. Edit window for "tarts"

Manager" below). On AT&T-OL, the *vi* editor is used. The File Manager window is ready for action again as soon as it resets its titlebar to non-busy state; it doesn't make you wait to finish editing before you can undertake new operations.

4.2.3 Moving or Copying Files

When you wish to move or copy a file, you can use the File Manager with OPEN LOOK's "drag and drop" operation. Drag-and-drop is a fancy term for "slide", and is familiar to us from a variety of similar operations. In the games of Chess and Checkers, for example, we select a player, and either lift or slide it to a new location, then release it. Computer drag-and-drop is similar: briefly, you select the icon(s) of the file(s) you want to move or copy, drag the icon into a window that represents a directory, and release the mouse button to "drop" the icon(s) into that directory.†

As an example, let's move the file *tarts*, to our home directory. First, to select a single file, *tarts*, just click **and hold**, SELECT with the pointer over the file's icon. Now if you move the mouse, you will see (Figure 4-5) that an outline of the file's icon follows the pointer:

Now that you have the icon "in tow", you can move it to any directory icon to drop it. At this point you don't have a second File Manager running to drop the file into. However, the File Manager that you are running shows icons for all the directories between the root directory ("/") and your current working directory (which in this example is */home/darian/ian/book/ol/filemgr/recipes*). So we'll move the *tarts*). file to the icon for

† This is one of OPEN LOOK's significant extensions beyond what the underlying X Window System provides. Hence it will not work correctly if for some reason you are not using the OPEN LOOK Window Manager; it will not work with *twm*, *mwm*, or *uwm*.

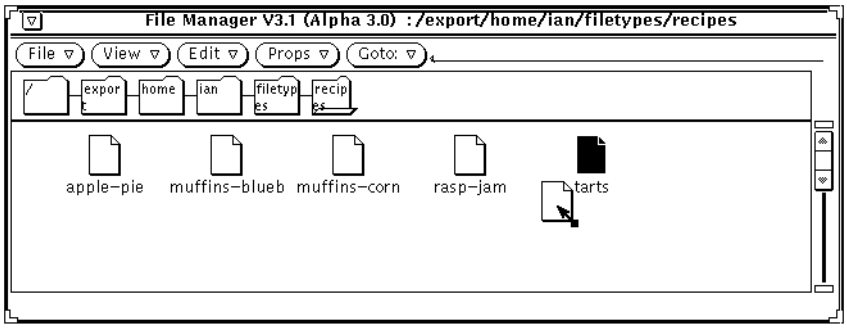


Figure 4-5. Icon outline for drag-and-drop

/home/darian/ian, my home directory, and drop the *tarts*, icon on top of my home directory's icon:

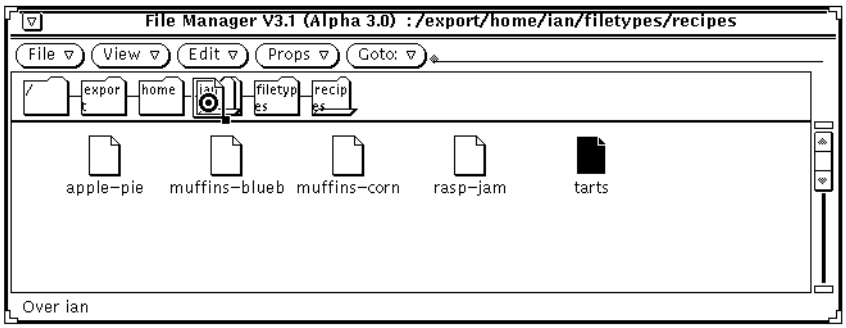


Figure 4-6. Moving the icon to drop it on a directory

Finally the File Manager screen will be updated to show that the file *tarts* is gone, and the window footer will contain the confirming message `Move to '/home/darian/ian' succeeded`, as shown in Figure 4-7:

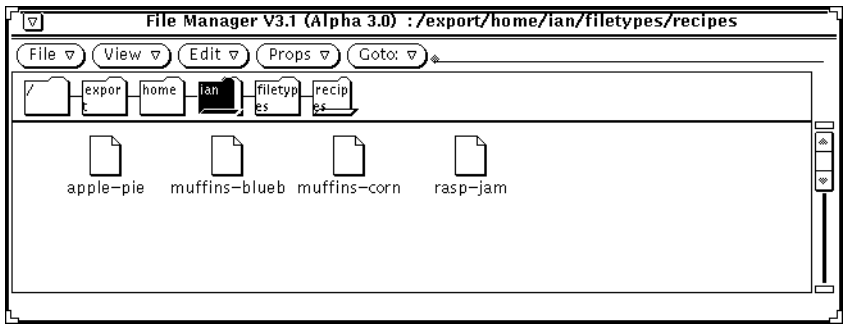


Figure 4-7. Double Confirmation: Icon gone, message appears

The move operation has been completed successfully.

To copy a file instead of moving it, all you do is hold down the DUPLICATE key (normally the Control or CONTROL key) while you click and hold the SELECT mouse button. The cursor shows slightly differently (see Appendix D, *Standard Cursors - X11 and OPEN LOOK*) to let you know that you are copying, rather than moving. Everything else is the same.

If you try to move or copy a file into a directory that is "read only" for some reason (for example, because you lack write permission on the directory, or the filesystem is mounted read-only), its icon will display a padlock icon while you hold the icon of the file you are moving over it.

On modern versions of UNIX (that is, SunOS, other BSD derivatives, and System V Release 4), you can move a directory as easily as a file, provided only that you can't move a directory from one "filesystem" or "slice" to another. On System V Releases 2 and 3, and System V Release 4 with the System V Filesystem Type, you cannot move a directory; you can only rename it within the same parent directory.

If you need to move several files, you may want to start another copy of the File Manager, and change it to the target directory. This will let you see what files exist in both directories before and after each move. And in this case you can just drop the icon anywhere in the icon window for the target directory.

Another possibility is to select multiple icons. As before, click SELECT (and release it) to select the first, and click ADJUST to select each additional icon. Then, click *and hold* SELECT, and you will get a "stack of icons" outline, which you can move to the target directory and release.

4.2.4 Deleting Files

Deleting files is just a special case of moving files. When you start the File Manager, it creates a separate window labelled *WasteBasket*.† The window is iconified, but you can still "drop" files into it.

If you select a file, drag it into the WasteBasket, and drop it, it is removed from the directory it was in. What could be easier?

But unlike the normal UNIX command for removing a file, this operation does not physically remove the file until one of the following occurs:

- You exit the File Manager, in which case you are asked if it's OK to remove the file(s) in the WasteBasket, and you click on "Yes."
- You change your mind about removing the file, double-click on the WasteBasket folder to open it, and move the file back where it was!

† On Sun OpenWindows Version 2, the WasteBasket icon sometimes appears with no icon label or image when it first comes up, so it will appear as a blank icon. But it still functions correctly.

4.2.5 Renaming Files

You can change the name of any file or directory in one of two ways. The quickest way is as follows:

- Click SELECT on the filename; the File Manager will underline the name.
- Treat the underlined filename as a text field; click on it, and then you can type a new name. Or, you can clear part of the name by clicking SELECT and clearing from there to the beginning with Control-U and type a new name, or delete or insert characters at will.†
- Then just hit RETURN.

This method is also used to assign names to objects created using *Create Document*; or *Create Folder*. The longer way, which you'd only use if you wanted to change other properties at the same time, is by selecting *NewFolder* and using the *File Properties* menu described in the next section.

For example, if we wish to create a folder or directory named *mideast*. in our example directory, the File Manager window will look like this after we create the folder:

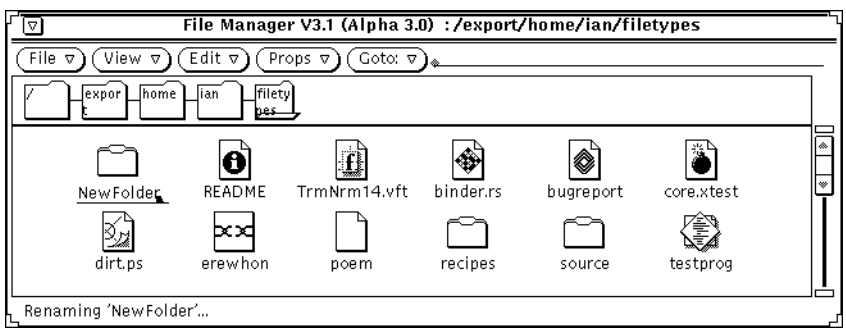


Figure 4-8. File Manager screen after Creating NewFolder

Now we need only click SELECT at the point we wish to change. Most commonly we want to erase the characters *NewFolder* completely and type a new name. So we click SELECT

† For a list of other editing keystrokes, see the discussion of *textedit* in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

at the right side of the existing name, type Control-U, and type the new name as shown in Figure 4-9:

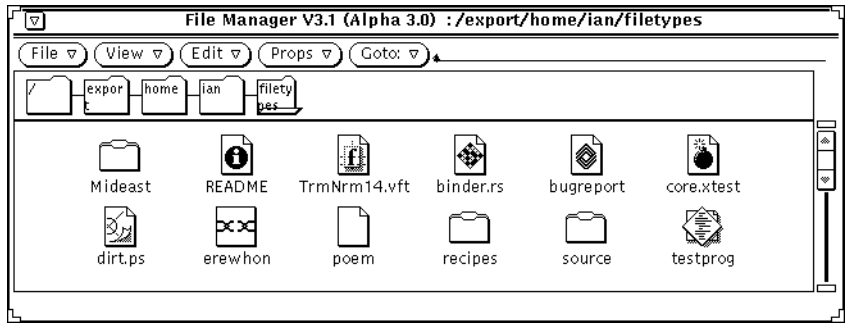


Figure 4-9. Directly renaming NewFolder to mideast

The rename will take effect as soon as you either hit RETURN or click SELECT elsewhere to take the focus away from the filename's text field.

Note that you can rename any object, not just a newly-created one, using this method. The trick is that you must first select the icon, *then select the filename* underneath the icon to mark it for renaming, then type the new name you want for it.

4.2.6 Displaying and Changing File Attributes (Properties)

This section demonstrates one of the clever ways in which the designers of OL worked to make it easy to use. To change any of the attributes of a file, you just call up its properties sheet. This replaces a whole grab-bag of operating-system-dependent utilities† with one simple, consistent menu. To display all the information about a file, simply select it, and

† On UNIX at least *ls -l*, *mv*, *chown*, *chgrp*, and the complexity of *chmod*.

click on the *Props* menu button (or drag it down to *File Properties...*). Figure 4-10 shows the properties sheet for a simple file.

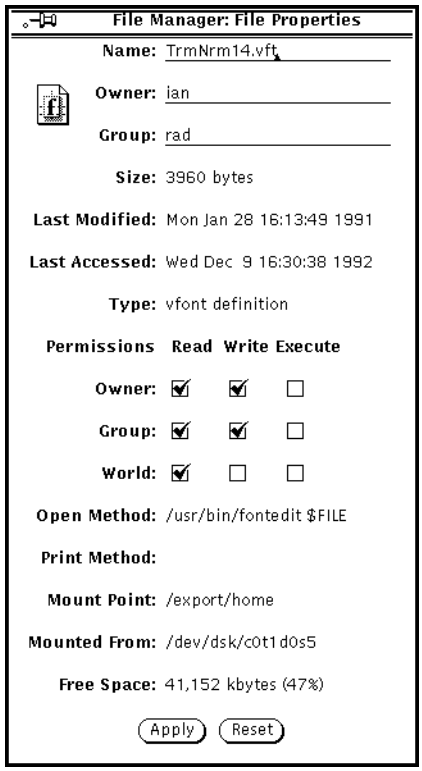


Figure 4-10. File Properties Sheet

If all you wanted to do was examine the file information, it's there. Read it, then with the pointer in this window press the Iconify button (L7), or select *Dismiss* from its window menu. If you want to change the attributes of the file, change them and click on the *Apply* button. To change the name of the file (rename it), you just type a new name in the "File name" text field, and click SELECT on the *Apply* button. To change the owner or group, type the name of a new group in the appropriate text field, and click APPLY. Of course to do this you must have sufficient privileges (own the files or be *root)*. To change permissions, click SELECT on the Checkboxes for read, write and execute for yourself, people in your group, and others on the system. At any rate, once you've specified any or all changes, you just click SELECT on the APPLY button to make them to take effect. If you have selected multiple files, then any properties that don't apply to all of the selected files are grayed out.

4.3 Menu Bar Operations

The Menu Bar, or row of buttons at the top of the File Manager window, gives you access to a large variety of operations. These buttons are named *File*, *View*, *Edit*, *Props*, and *Goto*. In this section we'll discuss those items that cause actions to be done for you: File, Edit,

File Props, and Home/Goto. We'll discuss *View* and *Tool Props* later, under the heading of *Customizing the File Manager*.

4.3.1 File Menu Operations

The *File* menu pulls down to offer you a choice of *Open*, *Print File*, *Create Folder*, *Create Document*, *Remote Copy...*, or *Your Commands*.

4.3.1.1 Open

The *Open* item is only active if a file is selected in one of the windows. This selection in turn pulls down to *File*, *With Goto arguments*, or *In Document Editor*. These let you open a file in various ways. *File*, which is the default if you just select *Open*, opens the file with whatever program is appropriate. A "worksheet" will be opened in Lotus 1-2-3; a Sun Raster file will be opened with *snapshot* program (described in Chapter 9, *Graphics Clients*), and so on. If you wish to pass command-line arguments to the given program, you can enter then in the *Goto* textfield and then use *Open* with *With Goto arguments* to pass the *Goto* field's contents as command line arguments. Lastly, if you wish to open the program using the default document editor (normally *textedit*), you can *Open* it *In Document Editor*, which overrides any program bindings for the file.

✗ Note that some types of files contain "binary data" and cannot be successfully edited with a text editor; if you get "garbage" characters in a file, it usually indicates that the file is binary, and you should not save it with a text editor or the file may be corrupted.

4.3.1.2 Print File

The *Print File* selection does just what it claims to: sends a copy of the selected file to your system's printer. The printer is normally your system default printer, but this can be overridden by use of the Property sheet, described below under the section *Customizing the File Manager*. Like *Open*, this selection is not available unless a file has been selected. In particular, you cannot select this item if a directory has been selected instead of a file!

4.3.1.3 Create Folder

Create Folder is like the UNIX or MS-DOS *mkdir* command in that it creates a new directory or folder. However, it does not prompt you for a name; it creates the folder and gives it the name *NewFolder*, to let you defer choosing a real name for it until you are ready. You then rename it in a manner similar to that described above in the section *Renaming Files*; for convenience, the filename text field is already selected, so you need only select the insertion point, type Control-U, and the new name.

On the other hand, if we want to change the permissions (or some other attribute) of the newly created directory at the same time as we rename it, we can use the *File Properties*; dialog, as follows:

- Click SELECT on *NewFolder*'s icon
- Pull down *Props* and select *File Properties*, or press the *Props* (L3) key. This will cause the *File Properties* dialog to appear.
- Change the *Name* text field from *NewFolder* to *mideast*.
- Change the permissions using the check boxes.

- The *File Properties* screen should now look something like this:

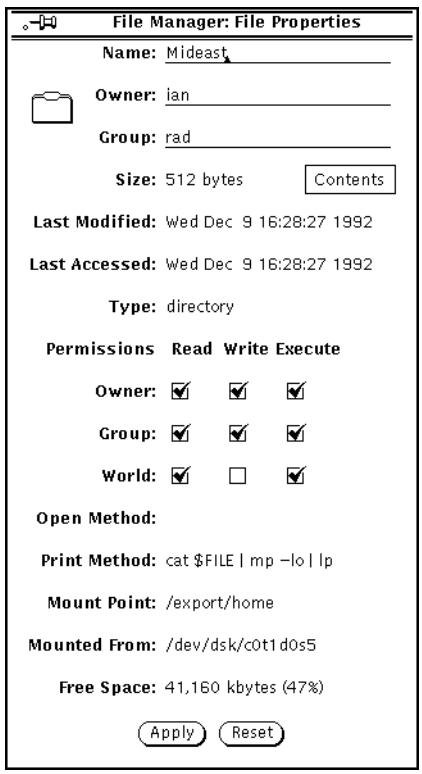


Figure 4-11. File Properties for renaming NewFolder to mideast

Click SELECT on the APPLY button; the change will be done and the *File Properties* dialog will pop down.

4.3.1.4 Create Document

How do you create a new document? Just select *Create Document*. The process is almost identical to the *Create Folder* operation just described. The new document is created as *NewDocument*, and you can edit it by clicking on its filename to set the text caret and using any text field editing, usually Control-U and type a new name. Once it's created, it behaves like any other file, except that it's created empty, so it has no contents or type. The most common operation next is to edit the new file to make it a text file.

4.3.1.5 Find

The *Find...* menu item pops up a dialog (see Figure 4-12). It can find files either by certain criteria (filename, owner, modification date, file type) or by contents.†

Let's say you wanted to look for all recently-modified text files containing the word "compendiate" Here is how you might set up the dialog, assuming that you used the extension *.txt*, for your text files:

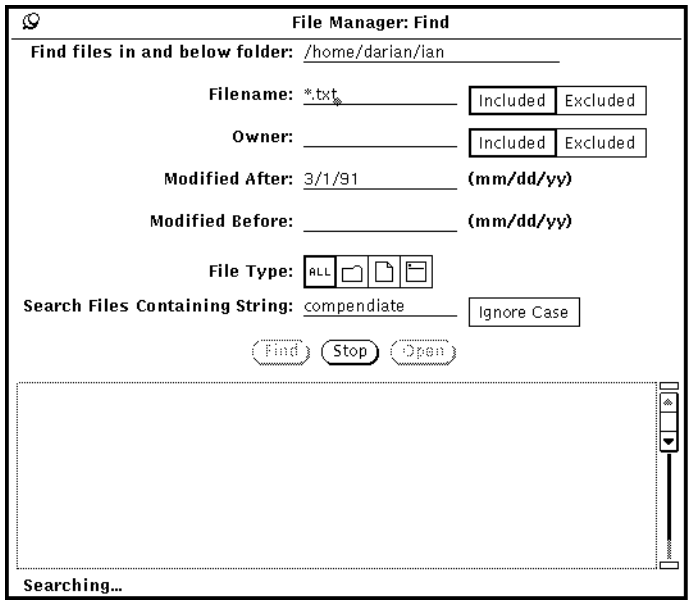


Figure 4-12. Looking for "compendiate"

Then click on *Find* to start the search. If any files are matched, they will be presented in the scrolling list in the bottom part of the window; these files can be clicked on, dragged, etc., just like other files in File Manager's display. This is shown in Figure 4-13.

4.3.1.6 Remote Copy

From time to time, you need to access a file that isn't on any machine you have a terminal emulator window on and that isn't exported for use as a network-mounted file. Then you must copy the file from the remote machine to your local system's disk. In command line mode, you would use the *rcp* utility. But in File Manager, you need only call up the *Remote Copy* selection. The *Remote Copy* operation allows you to copy file(s) to and from any machine that is reachable by the TCP/IP networking software and on which you have an account. You specify the source host and pathname, and the destination host and pathname. You can use either relative pathnames (path name does not begin with a '/' character) or absolute pathnames (path given does begin with a '/'). Relative pathnames are interpreted relative to your home directory on the remote machine, but relative to the current directory on the local machine (the machine that File Manager is running on). Let's copy a file from

† Experienced UNIX users will recognize this as similar to the *find* command in UNIX. If you need to perform a search that has more complexity than this *Find...*, dialog, you can use the standard *find* command. This function is in fact implemented using *find*. For example, when using the find function of File Manager to search in /usr/include for files containing the string TIOCCONS, we did a */usr/ucb/ps -a* in another window and saw this:

748 pts/0 S 0:00 find /usr/include -exec egrep -l TIOCCONS {} ; -follow

This way of finding strings in files is rather inefficient compared to "full text searching" as is done by SunSoft's *SearchIt* package or the contributed software package *LQ-Text*, among others.

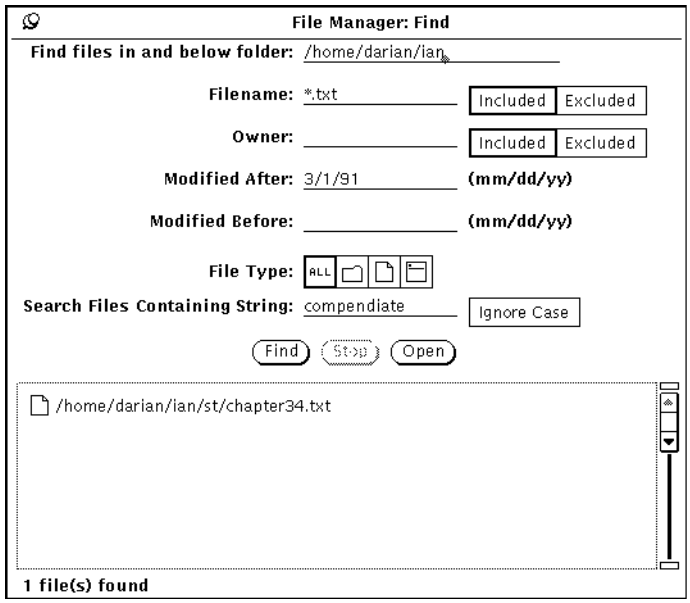


Figure 4-13. Found one file

machine *timbuktu*, to the machine we are on. Click SELECT on *Remote copy...*, and you get the pop-up window shown in Figure 4-14.

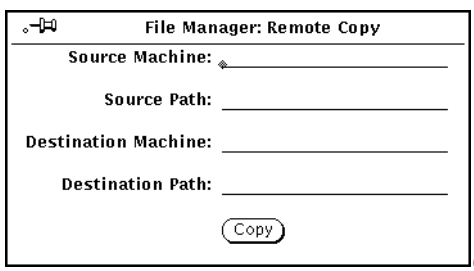


Figure 4-14. Remote copy pop-up window

The popup is pinnable, and since we might want to copy multiple files, let's pin it up. Then type the remote machine name, the filename, and a place to put it in the local directory, and select on the COPY button. It becomes busy:

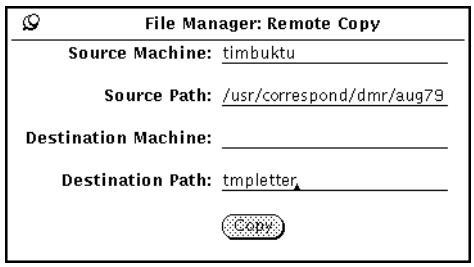


Figure 4-15. Remote copy window during copy

To copy out a file, just SELECT it in the File Manager window; the hostname and full path appear in the *Source Machine* and *Source Path* fields for you.

If there are any problems, such as "Permission denied", they will be printed in the left footer of the File Manager window, so watch that window for any error messages. If all goes well, the file you are copying to the local machine will appear in your File Manager view window after a short delay, assuming you're copying it to the same directory that the File Manager is in.

4.3.1.7 Your own commands

The *Custom Commands* menu lets you save a series of common commands, and execute them at the click of a button. This may appear to be redundant with the *.openwin-menu* of the OPEN LOOK Window Manager, but it is certainly easier to update. And it may make it easier to have different commands set up for different machines in a networked environment. To add a command to the list, just pull down *Add Command*.

You get a small dialog window with a text field in which to type a name for the command and the actual UNIX command line. When you've typed it the way you want it (see Figure 4-16), select *Apply*, and the new command will be added to your personal list. Next time you pull down the *Your Commands* menu, you will see the new command.

The other entry under this selection, *Shell*, starts up a terminal emulator window with an interactive shell running. This is rarely used, since you normally start shell windows from the *Programs*, menu of the *Workspace*, menu. However, it might be a good way to restart your window manager if it failed when you didn't have any shell windows running. Just select *Shell*, and when the shell window comes up, give the command

```
olwm &
```

and the window manager should reappear after a moment — you will see the titlebars reappear on your windows.

4.3.2 Edit Menu Operations

In most OPEN LOOK applications, the Edit menu is used to edit text. In File Manager, by contrast, it requests editing operations on whole files.

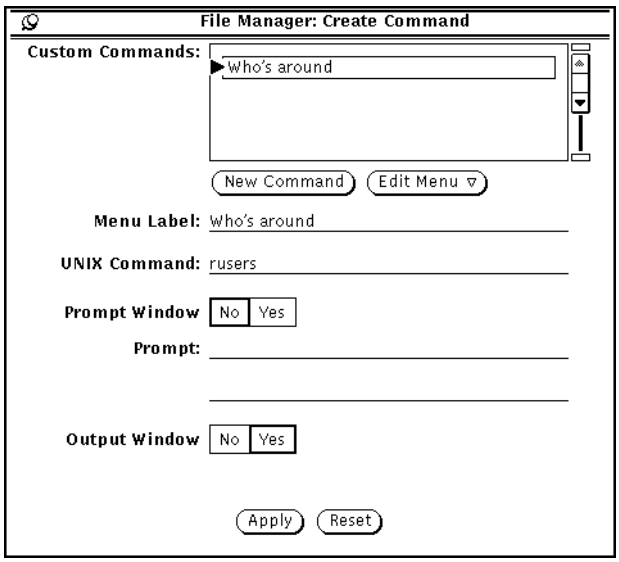


Figure 4-16. Create Commands Window

4.3.2.1 Select All

Select All, for example, selects (highlights) all the objects in the display window, so you can use them all as the target of some operation like print or move.

4.3.2.2 Link

The *Link*, selection allows you to create a *link*, to a file. A link is an additional name for the file; once you have created the link, you can refer to the file by either name. On Sun's OpenWindows, the new link is named 'filename.1' (or .2 if .1 already exists, and so on).

If your system supports "symbolic links", note that File Manager will create a regular link if the source and target directories are on the same filesystem, otherwise it will create a symbolic link. If you are on System V Release 2 or 3, you can ignore this distinction.

Symbolic links behave a little differently than regular links. When you add or remove a regular link, the file's link count is incremented or decremented. If the link count reaches zero, the file's data blocks are discarded by UNIX and the file ceases to exist. Symbolic or "soft" links are actually files that contain the real *name* of another file; when you "open" (to read or write) a file that is a symbolic link, the UNIX kernel reads the symbolic link and then tries to open the other file by name. A "symlink" can therefore do two things that no ordinary link can do:

- it can cross disk partition or "slice" boundaries.
- it can be used to point at directories.

The drawback of symlinks, however, is that if the file they point to is renamed or removed, the symlink still points to it; it will show up in an *ls* listing, for example, but you won't be

able to read it or write it, which is confusing. Such misleading links are called "broken" or "dangling" symbolic links, and are shown by File Manager with the "broken chain" icon used to represent the file "erewhon" in Figure 4-1.

4.3.2.3 Copy

Copy works just link *Link*, except that it copies the file instead of linking it. That is, after a *Link*, you have only one copy of the file on disk, accessible under two different names. But after a *Copy*, you have two different files (though they initially have the same contents), each with its own name.

4.3.2.4 Cut and Paste

Paste and *Cut* work together to move files into and out of the directory. Select a file and *cut* it, and Poof! It's gone! Where did it go? By default, it goes into the WasteBasket folder, just as if you had dragged it there and dropped it. But *Cut* can be *pulled right* (hold the MENU button and drag the pointer to the right to pop down a new menu) to produce a menu asking if you want to *Cut to Wastebasket*, (the default), *Move to Clipboard*, or *Really Delete*. The third item tells the operating system to unlink or remove or delete the file, while the middle item moves it to a special folder called "the Wastebasket," from which the file can be retrieved later by moving it into a (possibly different) directory. The *Show Clipboard* item shows what has been cut into the File Manager Wastebasket.

4.3.3 Goto Menu Operations

To help you move around quickly by directory name, the *Goto* menu lists your home directory and several of the other directories that you have visited most recently. Sliding the pointer down this menu and releasing on any one of these will make it the current directory. Initially, the default is your home directory, so you can get to your home directory just by clicking SELECT on the *Goto* selection item.

*But n*otice that there is a text field to the right of the button. If you type a path name in this text field, then click SELECT on the *Goto* button, that directory will be made your current directory. If you click MENU on the button, you will see two items. One is the default, the contents of the text field (or as much of it as will fit on the display).

This field has another use, too. You can use it to specify a range of files to operate on. This has the effect of "wildcards" in the UNIX shell environment. For example, to select all the PostScript files in a directory, just type the string **.ps* in the *Goto* field and then press Return or Enter, and all the files that match will be marked selected. Then if you want to delete them all, just hit the CUT key (L10). Or, drag any one of the selected icons onto the trash can (or any other drop target, such as print!), and *all* the files will be moved or copied as appropriate.

In fact, you don't even need to use this field. Just move the pointer into the file icon display pane and type your pattern there. The pattern will appear in the footer as you type it, and as each character is typed, any files that match will be selected. This is faster, but more exacting, as you don't have the textfield's editing abilities at your disposal.

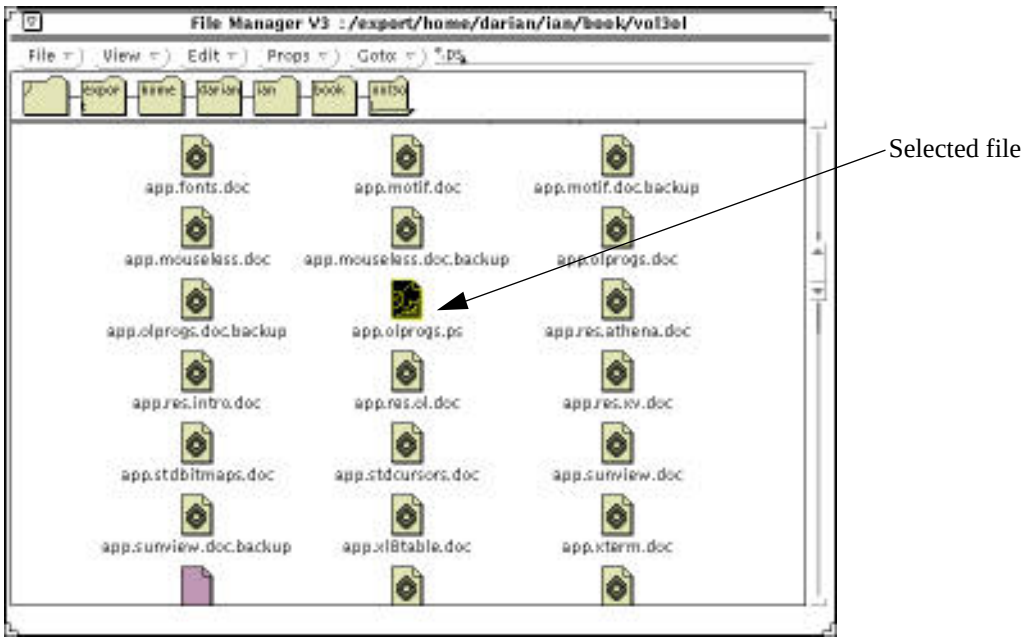


Figure 4-17. Wildcard selection in File Manager

4.3.4 MENU button in the display

As a convenience, there is a menu attached to the MENU button while in the File Manager window that contains some of the most common items of the ones we've seen. The list includes *Select All*, *Cut*, *Copy*, *Paste*, *Delete*, *Print*, and one called simply *Up*. Except for the last one, these all do the same thing as they do when activated from the top-level buttons in the menu bar. The last one does what you might expect from its name; it moves you up one directory. It is inactive when you are in the root directory ("/"), since there is nothing above that directory.

4.3.5 MENU button in the Wastebasket folder

If you open the *Wastebasket* icon, you see a folder display for items that have been moved or cut into the wastebasket. There are no menu buttons, but the MENU pointer button has a menu similar to the menu just described. It has one important addition; the first item is *Empty Wastebasket*, which actually deletes the files in the wastebasket—files so removed are *really gone!* This is useful when your hard disk is running low on space

4.4 Customizing the File Manager

The overall behavior of the File Manager is fixed. But you can change the details of its behavior at several levels. First are the *View*. controls on the top row. Second is the Properties sheet. The third level of customization is in configuration files.

4.4.1 View Menu Button

As we've seen, the menu bar items are *File*, *View*, *Edit*, *Props*, and *Goto*. Most of these were discussed above. The File Manager *View* menu items control what you see, or your view of the files. Unlike many of the other buttons, changes you make here are saved in a file for later File Manager sessions, that is, they stay in effect until you change them again.

The first item is either *Path* or *Tree*, initially *Path*. When the setting is *Path*, the "parent directories" window shows a single row of icons for the path to the current directory. When the setting is *Tree*, however, you get a visual presentation of the parent directories and all the parallel directories, as shown in Figure 4-18

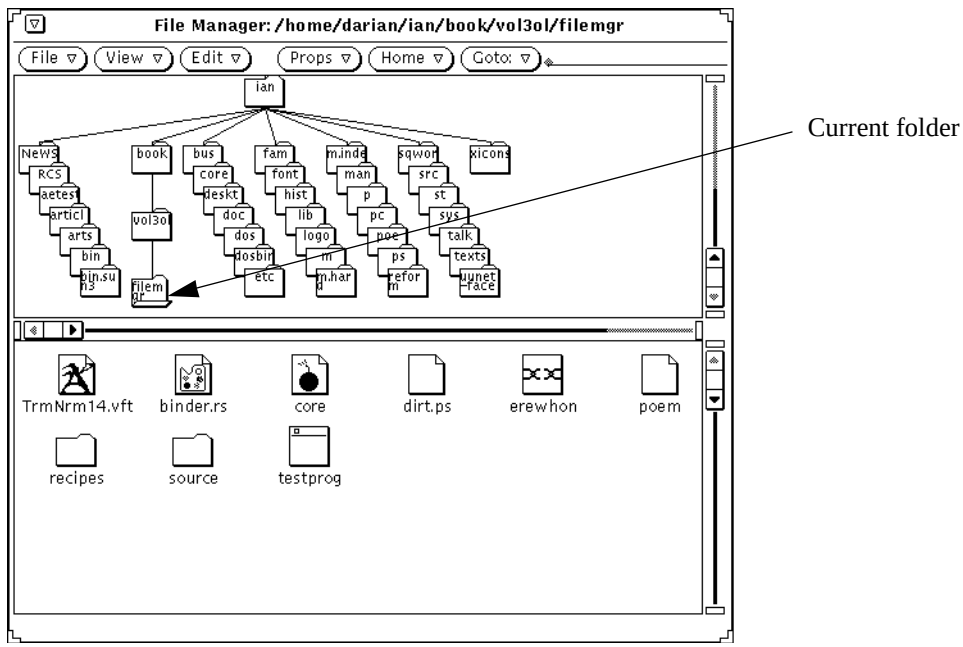


Figure 4-18. View set to Tree shows the directory tree

Now it may happen that the directory hierarchy that is shown on the screen gets large and complicated, if you have lots of subdirectories. To help you remember where you are, the currently-open folder is drawn with an "open folder" icon (see Figure 4-18) to remind you of your place in the overall directory hierarchy.

Each icon in this window is "active." That is, you can double-click on it, drop a selection into it, etc., in any way that makes sense. The next few items, *Hide Subfolders*, *Show All Subfolders*, *Begin Tree Here*, and *Add Tree's Parent*, are used to control the tree display, but only if the mode is *Tree* **and** you have selected at least one directory with subdirectories (folder with subfolders). *Hide Subfolders* and *Show All Subfolders* control whether subdirectories of the current (or selected) folder will be shown. Note that selecting *Show All Subfolders* is a recursive operation. That is, it shows all the subdirectories, sub-sub-

directories, sub-sub-sub-directories, and so on. For this reason, the left footer message is set to “This operation may take some time” while the subdirectories are being found. But you can get some fantastic displays of your system’s directory tree. Try this: before going to lunch, and running File Manager on your own workstation, select the root directory, and click on *Show All Subfolders*. When you get back, there will be vertical and horizontal scrollbars for the view window, and you will be able to see the entire UNIX filesystem just by moving the scrollbars. Of course, you can also resize the window wider to see more of the display at one time:

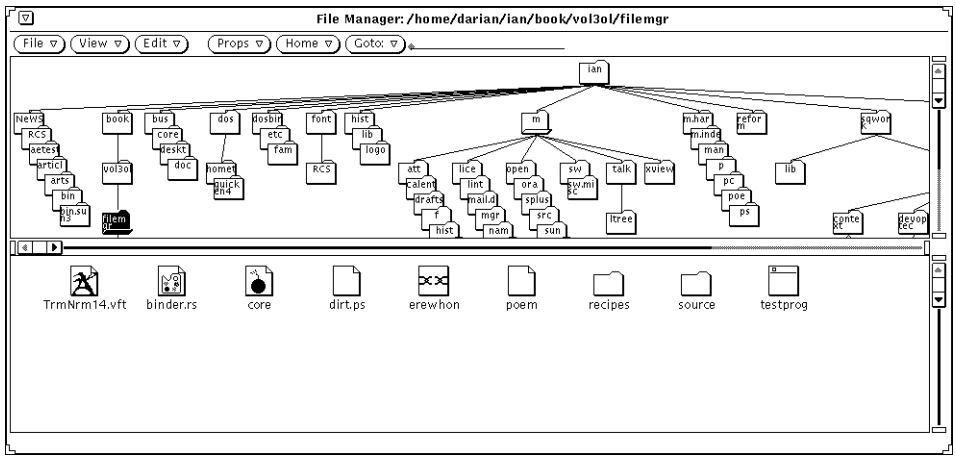


Figure 4-19. View Tree made wider

Begin Tree Here and *Add Tree’s Parent* are used to truncate the tree (so that everything above the selected directory is removed from the top of the display) and to extend it upward by one directory level, respectively.

The menu has several choices to specify how files and directories are to be sorted and displayed. The types are either *List* or *Icon. Icon*, which is the default, uses the large file and directory icons shown throughout this chapter. *List* uses smaller icons like those shown in Figure 4-20. The *Icon* display can be sorted by name or by “type”, which sorts all entries of a given type together. The *List* display can be sorted by name, type, size (largest first), or by modification date.

These smaller icons show less information than the larger ones; they only identify the object as “file,” “folder” or “other.” (Compare *erewhon*’s look here with its look previously.) However, the small icons allow you to show more files in a given space (useful in a large directory), and you can still select them, drag-and-drop them, etc., just as you can the full-size icons.

Which format you choose — large or small, sorted by name, or by type, or by date — is completely up to you. It’s a matter of taste. Try them out from time to time, and see which makes most sense to you.

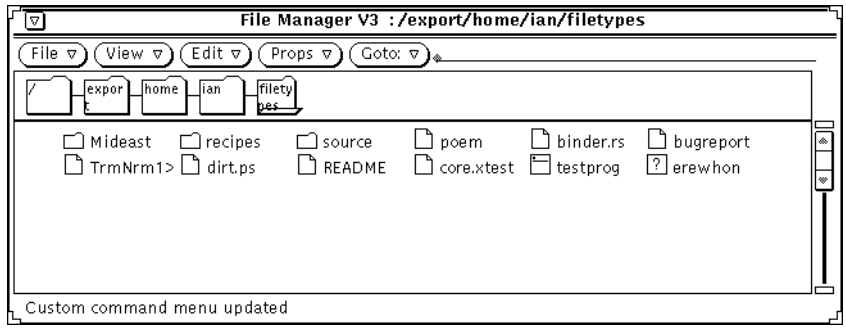


Figure 4-20. Folder Display set to List

4.4.2 The View/Customize Properties Sheet

There is a "Properties"-like window under the *View* menu, named *Customize View* (See Figure 4-21). Many of its items are similar to those of the *View* menu discussed previously,

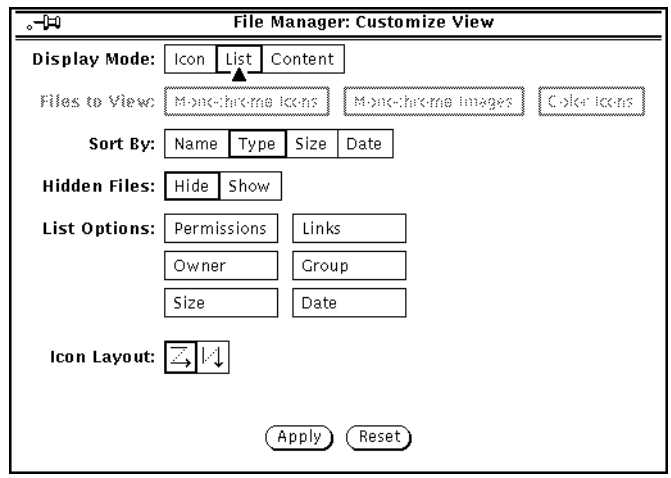


Figure 4-21. The Customize View Window

but the window also lets you control whether hidden files ("dot files", whose name begins with a period or dot) are displayed, and whether icons are layout out in rows across the screen or in columns down the screen.

You can also select *Content* as the Display Mode setting. This causes raster and icon files in known (Sun) formats to be displayed as themselves, at a slight cost in speed. However, at least in Version 3.0, this display is garbled if the raster files are of odd sizes.

If you select *List* as your display mode (either on the Customize View window or in the View menu), the Customize View window lets you control the display of the information equivalent to that of the UNIX command *ls -l*. You can enable or disable individually the viewing of permissions, size, owner, group, link count, and modification date. Figure 4-22 shows the display of just permissions and file size, along with the names which are always displayed.

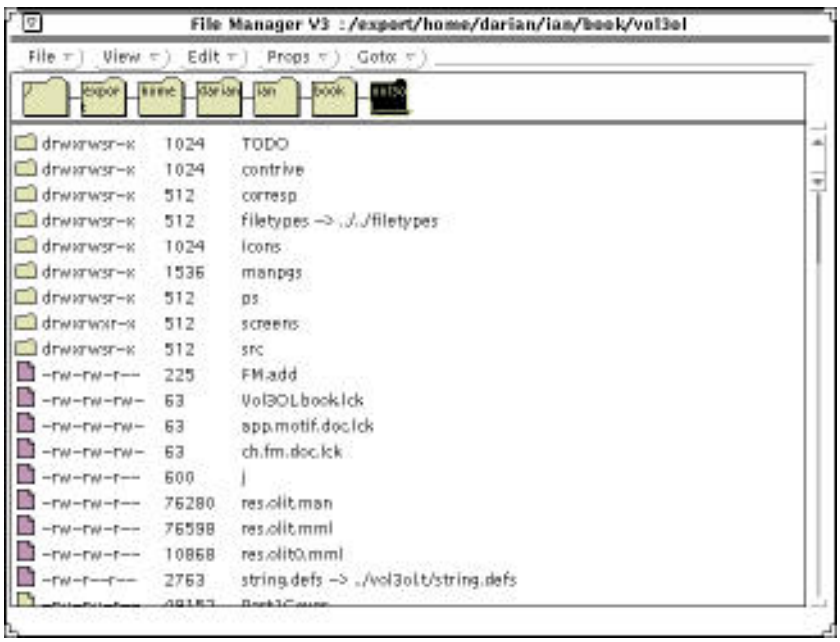


Figure 4-22. Folder display with permissions and size only.

4.4.3 The Tool Properties Sheet

You display the Properties sheet for the File Manager by pulling down the *Props* top-level menu and releasing on *File Manager*.The resulting screen looks like Figure 4-23 on OpenWindows.This gives you control over the optional features of the File Manager's behavior:

- the default print command
- something called the *View Filter*, which allows you to only look at certain filenames, for example, "*.*txt.*"
- the longest filename to be displayed
- whether deleted files are kept in the *Wastebasket* folder, or actually unlinked
- selecting the default editor for documents.

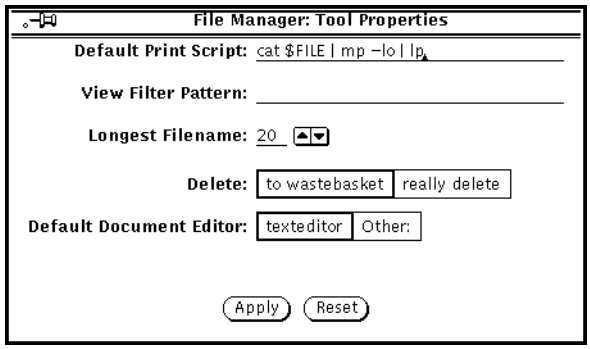


Figure 4-23. The File Manager Property Sheet

4.4.4 Customizing File Bindings

The third level of customization lets you control the way File Manager reacts to what it fins in your directories. Many of the responses are controlled by a series of configuration files. Unfortunately, the format of these configuration files does depend on which implementation you are using, AT&T's or Sun's.

4.4.4.1 Binder: Customizing the File Manager in OpenWindows

The OpenWindows version of File Manager uses a package called the *classing engine* to control the display of different file types. The Classing Engine is used by File Manager (and other DeskSet programs), as well as the OPEN LOOK Window Manager, to classify files. There are three sets of classing engine files; one "network wide" one for all files served by a given NFS server, one "system-wide" file (in the */etc* directory on each machine), and one personal one (in your home directory).

Now updating these files is not for everybody. You only need to do this if you have a new type of file that the Classing Engine doesn't know about. Some vendors' installation software, for example, updates the files automatically, without running the binder. There is a command-line interface to the Classing Engine files; see the man pages for *ce_db_merge* in OpenWindows for details.

You normally update the Classing Engine files using a program called *binder*. Once you've started *binder*, you should see something like Figure 4-24 in the *binder* dialog box.

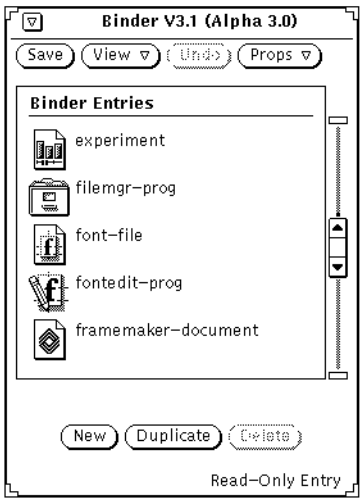


Figure 4-24. Binder dialog

The initial screen is a scrolling list of icons (the same ones used in the File Manager display) and the associated file type. Once you have selected one, you can customize either its icon properties or the filetype properties. Figure 4-25 shows using the *Icon* properties to change the color that a given icon type is displayed in.

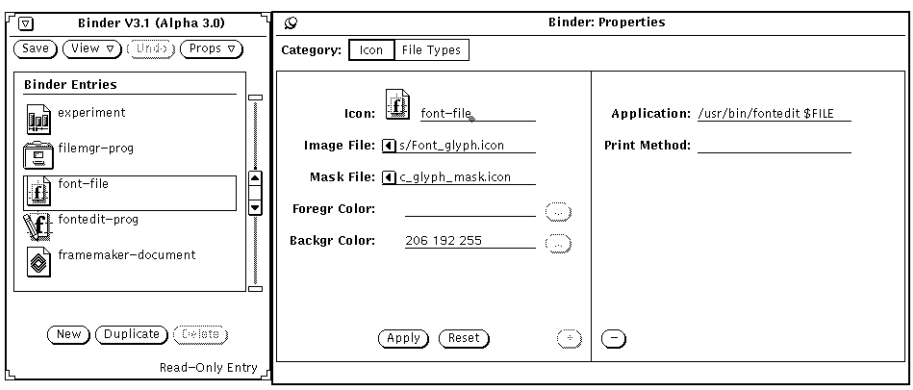


Figure 4-25. Binder Icon Properties Dialog

Users of OpenWindows Version 2 should note that the file layout is totally different, but use of the *binder* program is similar to that described here†

† .There is a bug in the OpenWindows 2.0 version that you should know about if you've read this far. If your file refers to an XBM file rather than an icon-format file, then *olwm* will still process it, but *binder* will reject the entry with a message like "icon file /home/darian/ian/xicons/xterm parse failure" *and will delete the entry if you save your file*.

Whether you update the system-wide file or your own personal file, you must restart the File Manager and the OPEN LOOK Window Manager before your changes will affect these programs.

We've now shown how to use the File Manager to navigate your directory hierarchy, to select files and operate on them in various ways, and showed several different methods of customizing this versatile and valuable tool. The next chapter discusses the terminal emulator and text editing under OpenWindows, after which we turn our attention to the OPEN LOOK Window Manager.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 5

The Cmdtool/Shelltool Terminal Emulator

There are many "terminal emulator" programs available for The X Window System. This chapter shows you how to use the OpenWindows standard terminal emulator, a program known as *cmdtool*. *Cmdtool* is a general-purpose terminal emulator with a variety of features such as scrolling, logging, history editing, and so on. And unlike some other programs, such as the MIT version of *xterm*, *cmdtool* is compliant with the OPEN LOOK specification. and is therefore very similar to the editing windows provided by most OPEN LOOK-conforming applications. The name *shelltool* invokes the same program in a different mode.

This chapter also discusses the OpenWindows text editor, *textedit*, since it shares many features with *cmdtool*†.

Like any other terminal emulator such as *xterm*, these tools provide a window that works much like a standard computer terminal, and that lets you do anything you can do with a standard terminal. Each such window is connected to a *shell* or command interpreter program (normally the same shell that you use for your login shell), so you can run shell commands directly. Once you have one terminal emulator window on your screen, you can use it to start up any other clients you need, including other terminal emulators. You can run as many terminal emulators as you need. For example, you might be examining or editing a program's source code in one *cmdtool* window, while compiling or debugging it in another.

The easiest way to start a new *cmdtool* is from the *Workspace->Programs* menu. As it is the default "Program," you need only click SELECT on the *Programs* menu to start an additional *cmdtool* at any time.

† If you aren't using OpenWindows and wish to get a copy of *cmdtool*, you can FTP it (along with the XView toolkit, which is required to compile *cmdtool*) from MIT; see the Preface for details.

When you start a *cmdtool* process on the command line from an existing *cmdtool* window, the second *cmdtool* inherits the "environment variables" of the first, including the setting of the DISPLAY variable. The program running in the second also begins in the current directory of the first shell. Otherwise, there is no real relationship between the two windows.

The basic operation of *cmdtool* will be familiar to anyone who has used an interactive computer terminal; you should be able to work productively immediately. You type a command, hit return, and see the results. As well, terminal emulators aren't limited to the fixed size of twenty-four lines by eighty characters; you can make them any size.

But *cmdtool* is more than just a "dumb terminal." Two useful features of most modern terminal emulators are an optional scrollbar to review text that has gone "off the top" of the display, and a "cut and paste" facility that lets you rearrange text or even copy it from one terminal emulator window into another. *Cmdtool* includes these, but also has comprehensive text editing features built in.

The *shelltool* and *cmdtool* programs have been part of Sun's graphics offerings almost as long as there have been Sun Workstations. The SunView™ window system used under SunOS Release 3 provided these programs as its standard terminal emulators for several years before the X Window System was even invented. It is not surprising, then, that these two programs offer the same kind of functionality as does the MIT X terminal emulator *xterm* but in different ways.

In those days, you had to remember to use *cmdtool* for full-screen programs such as *vi*, and use *shelltool* for programs that used a simple tty-style interface (there was also *gfxtool* to start up certain types of graphics clients). Under OpenWindows, these two terminal emulators are in fact just two names ("links") for the same program, so for convenience I'll simply use the term *cmdtool* to refer to both of them. The most significant difference between them is that *cmdtool* starts up with scrolling on, and *shelltool* comes up with scrolling off.

The main philosophical difference between *cmdtool* and the MIT terminal emulator *xterm* discussed in Appendix A, *The xterm/olterm Terminal Emulator* is that *cmdtool* tries to let you do full editing on your terminal's history, while *xterm* preserves the history intact and only lets you copy from it, and paste words or lines only into the current input buffer. *Cmdtool* thus carries the notion of "putting shell history into the terminal" one step further than does *xterm*, at a slight cost in efficiency. In fact, *cmdtool* has built into it a complete text editor, which can be used to edit either the shell history or separate text files. This editing facility can also be started up as an editor, called *textedit*; we will discuss this editor here, because it is so similar to *cmdtool*. Another difference is that *xterm* tends to be a little faster than the current version of *cmdtool* at inserting characters, which may make it preferable for touch typists and others who type very rapidly.

5.1 Terminal Emulation and the sun-cmd Terminal Type

If you've used more than one kind of terminal or terminal emulator, you probably realize that they don't all behave the same way. In particular, terminals differ in how they respond to *escape sequences* or series of special characters typed by you or sent from the computer to the terminal. One terminal might clear its screen upon receipt of a Control-X character, for example, while another might need the four-character sequence <ESC>H<ESC>J in order to clear the screen, where <ESC> means the ASCII control character "Escape" or just "ESC". As a result, it can be very hard to write computer software to drive all the different terminals in the world. The computer industry has taken two approaches in trying to tangle with this problem. On the one hand, an ANSI Standards Committee has drawn up a standard, based largely on the DEC VT100 terminal, to specify how terminals should react to escape sequences. And many manufacturers (and writers of terminal emulators) are using this standard.

On the other hand, there are still many terminals in widespread use that do not conform to this standard, and an equal number of variations among those that claim to. As a result, UNIX systems include one of two similar facilities, called "Termcap" and "Terminfo", which are databases of the "capabilities" of all the terminal types, or the "info" needed to drive each type of terminal. Termcap/Terminfo provide information on literally hundreds of different brands of terminals, and are used by screen editors such as *vi* and *emacs* to determine how to drive the particular terminal you are using.

But the trick is that you must somehow tell the software what type of terminal you are on. If you omit this, or get it wrong, command-line commands such as *ls* and *cat* will still work, but screen editors and paginators—any screen-oriented software—will mess up.

How do you tell the system what kind of terminal you have? For each type, there is a name. The name *sun-cmd* is used for the *cmdtool* terminal emulator. The name *xterm* refers to the *xterm* terminal emulator discussed in Appendix A, *The xterm/olterm Terminal Emulator*. Both the *cmdtool/shelltool* and the *xterm* terminal emulators normally set their own TERM environment variable. You should be careful not to override this terminal type in your shell initialization files. For details, please refer to Section 3.5, "Customizing your Session Start-up" on Page 72, and the *cmdtool* reference page in Part Three of this Guide.

Additionally, *cmdtool* interprets a variety of escape sequences of its own. These allow you to resize the window, change the contents of the titlebar, etc. The escape sequences are described in Appendix E, *Control Sequences for xterm and cmdtool*.

5.2 Resizing a cmdtool Window

The X Window System and the OPEN LOOK GUI allow you to change the size of almost any window at almost any time. Most X Window System programs respond reasonably well to having their size changed. However, when a terminal emulator is resized, there are implications for any screen-based clients it is running at the time. The bottom line is that it's OK to resize a *cmdtool* window, but not *while* it is running a full-screen program such

as a screen editor or word processor. Most OpenWindows clients *other* than terminal emulators can be resized at any time. A real terminal normally cannot be resized (except with a steam roller), so some screen editors get upset when the terminal is resized. However, several means have been worked out to solve this problem. First, whenever an editor such as *vi* or *emacs* starts up using the *sun-cmd* terminal type, it asks the terminal emulator for its current size, so it doesn't matter whether you have resized it or not. The *resize* command described in the previous chapter is not needed and in fact does not work with terminal emulators other than *xterm*.

However, a problem comes up if you resize a window *while running* a screen editor. Some versions of UNIX allow such a program to be notified (by a signal named SIGWINCH) when its window (terminal) size changes. However, the SunOS 4.1.x version of the *vi* screen editor apparently is derived from the System V *vi* rather than the current Berkeley version, and does not respond to SIGWINCH. Therefore, do not resize a terminal emulator *while* a screen editor is running in a *cmdtool* or *shelltool* window. †

As mentioned in the previous section, you can resize a *cmdtool* dynamically using escape sequences. Example 5-1 is a short shell script that does so.

```
#!/bin/csh -f

# Simplified from a script from Jay Plett jay@silence.princeton.nj.us>
#
switch ( $#argv )
    case 0:                 # no args, set default
        set columns = 132
        set rows = 34
        breaksw
    case 2:                 # two aregs: rows cols eg 24 80
        set columns = $2
        set rows = $1
        breaksw
    default:                # else fall through
        echo Usage: `basename $0` '[rows cols]'
        exit 1
endsw

if ( $term == sun-cmd ) then
    @ cols = $columns + 4
else
    set cols = $columns
endif

stty rows $rows columns $columns# tell unix new term size
```

† The *vi* sequence

```
Q
:set term=sun-cmd
:vi
```

will usually fix it if you accidentally resize a window running *vi*.

```
echo -n "^[[8;$rows;${cols}t"   # tell cmdtool (N.B. "^[" is ESC)
```

Example 5-1. C Shell Script to resize cmdtool

If you install this script in your *bin* directory as *cmd-resize* and mark it executable, you can give the command

```
% cmd-resize 30 90
```

to make your current *cmdtool* window be thirty lines deep by ninety columns. If you make the window smaller than its current size, press the *End* key or Ctrl-Return to move to the bottom of the current window.

While we're on the subject of useful scripts for *cmdtool*, here are two more. The first one sets the title, and the second sets the icon label.

```
#!/bin/sh
# cmd-title - set the title in a cmdtool window
echo -n "^[]l$*\\"

#!/bin/sh
# cmd-iconlab - set the icon label in a cmdtool
echo -n "^[]L$*\\"
```

These use the *ucb* version of the *echo* program; if the *-n* appears on your screen when you run them, set */usr/ucb* to the front of the search path in the script.

5.3 The cmdtool Menus

Cmdtool's menu layout is simpler than that of *xterm*; there are two main menus, depending on which mode you are in, and a greater reliance on X resources and on initialization files than is the case with *xterm*. When you start the program in *cmdtool* mode, you get the *cmdtool Term Pane* menu; when you start it in *shelltool* mode (or start it as *cmdtool* but are running a full-screen application such as *vi*), you get the *shelltool Term Pane* Menu.

5.3.1 The Cmdtool Term Pane Menu

The *cmdtool Term Pane* menu has six selections: *History*, *Edit*, *Find*, *Extras*, *File Editor*, and *Scrolling*. Most of these will be described below, in the section on Editing. The *Scrolling* selection is an exclusive choice: on or off. This item's name belies its effect: when you are in *cmdtool* mode, scrolling is on. When you turn scrolling off here, poof! You are now in *shelltool* mode, and the main menu is now the one listed for *shelltool*.

5.3.2 The Shelltool Term Pane Menu

This menu has only four choices: *Enable Page Mode*, *Copy*, *Paste*, and *Enable Scrolling.*

The *Enable Page Mode* causes pausing after every screen full of text, rather like the convenience of having *more* or *pg* built into your terminal so that it benefits every program. However, this mode is disabled if you go into *cmdtool* mode.

The *Copy* and *Paste* selections are the same as those in *cmdtool*'s *Edit* sub-menu, and likewise behave as mentioned below under Editing.

The *Enable Scrolling* item turns on the scrollbar feature. This has the side effect of flipping you back into *cmdtool* mode.

5.4 Using the OPEN LOOK Scrollbar

Many OPEN LOOK programs, including *cmdtool*, allow you to view on the screen one part of a larger "virtual screen." The OPEN LOOK mechanism that controls this is called a *scrollbar*, and allows you to move the on-screen window up or down either one item (such as a line of text) at a time, or smoothly drag it up or down. Unlike most other GUIs, the OPEN LOOK scrollbar also allows some other functions, such as splitting the view of the larger background into several smaller, independent on-screen views.

Note that *cmdtool* normally includes a scrollbar, while *shelltool* does not, unless you have enabled it from the window menu. You can have your OPEN LOOK scrollbars placed on the left side of your applications or the right side, either by using the Workspace Manager (see Chapter 6, *Using the OPEN LOOK Window Manager*) or by specifying an X resource variable (see Chapter 12, *Setting Resources*).

There are two other types of scrollbars you are likely to run into while using The X Window System. The Athena Widget scrollbar is described in Appendix A, *The xterm/olterm Terminal Emulator*. The Motif scrollbar is described in Appendix M, *OPEN LOOK and Motif*. In this section, we will concentrate on the OPEN LOOK scrollbar. Once you've learned it, the others can be learned easily.

First, let's review the parts of the scrollbar, as shown in Figure 5-1Note that it looks much like an elevator riding up and down on an elevator cable. And the position of the elevator within the overall scrollbar shows where you are, proportionally, in the text or list that you are scrolling; if the elevator is at the top anchor, you are seeing the top of the text or list; if it's halfway between the anchors you are seeing the middle, and so on.

The most common use of the scrollbar is by clicking SELECT on the elevator. Pressing and holding SELECT on the center box (the drag area) lets you slide the scrollbar up or down at will. Clicking SELECT on the up arrow will move the elevator up; clicking it on the down arrow will move the elevator down. As the elevator moves up, the part of the text or list that you see moves up; in other words, you move closer to the top of the file. However, if you've never used a computer window system with scrollbars before, don't be thrown off by what happens; when you click on the up arrow, the text on the screen moves down! This is just a consequence of moving the view window upwards, and is analogous to the fact that, looking out the side window of a car driving forward, the scenery appears to move toward the back of the car. This is so obvious that you never think about it, once you've seen it a few times.

Note that when you click on either arrow, the pointer jumps† when the elevator moves, to stay on the arrow. Thus you can click SELECT repeatedly to move several lines at a time without having to move the pointer each time. In fact, if you hold down SELECT you will

† You can disable this in the *Properties* editor.

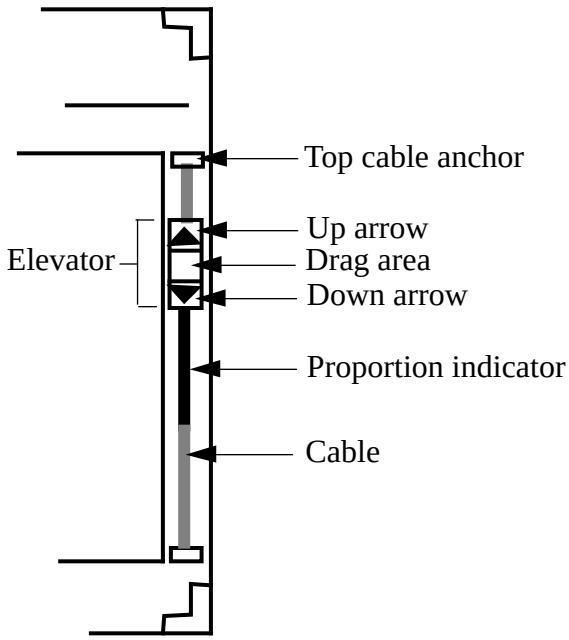


Figure 5-1. Parts of the OPEN LOOK scrollbar

notice that it "auto-repeats" like the keys on your keyboard: it keeps moving along until you reach the end of the data.

You can also click on either cable anchor to return the elevator to that end. Clicking on the top cable anchor moves you to the top of the file; clicking on the bottom anchor moves you to the bottom.

What's in between the elevator and the cable anchors? The cable, of course. You can click SELECT on any point of the cable. When you do, the elevator moves up or down by one screen full of text each time and again, the pointer jumps if necessary.

You may have noticed that one portion of the cable is thicker than the rest of it, or rather is black while the rest is gray. The black part is the so-called *proportion indicator* that shows how big the on-screen portion of data is in *proportion* to the total amount of data viewable. The proportion indicator is always attached to the elevator. If the elevator is at the top of the column, the proportion indicator will be below it; if at the bottom, above it. If the elevator is somewhere in the middle, the proportion indicator will be partly above and partly below, in proportion to where you are in the total data. If the data is so huge that the proportion indicator would be invisible beneath the elevator, a tiny proportion indicator is used. If all the data is visible, as when you start up a *cmdtool* program, the entire cable is black.

5.4.1 Jumping with the scrollbar menu

The scrollbar has its own menu, which you can get at any time by moving the pointer into the scrollbar and pressing the MENU button. There are usually four items on it: Here to top; Top to here; Previous, and Split View. These simply allow you to move the top line that is on the screen ("top") to where the pointer was pressed ("here"), to move the lines where the pointer was pressed to the top of the screen, and to undo either of those two move operations (the fourth item, *Split Views*, is optional, at the discretion of the application writer; we'll discuss it later). For example, suppose that we have this window:

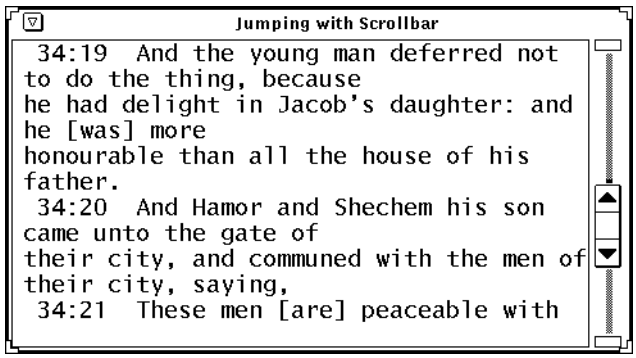


Figure 5-2. Moving top to here: before

and want to scroll it so that the first line ("34:19 And the young man...") is near the bottom. We just move the pointer into the scrollbar near the bottom, click MENU, and release on *Top to here* to move the top lines to where we are, as shown in Figure 5-3

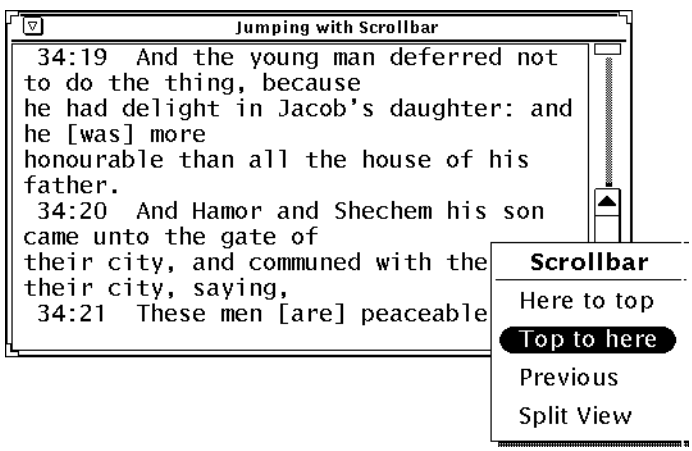


Figure 5-3. Moving top to here: before

After we release on *Top to here*, the top line will be at the bottom, as in Figure 5-4

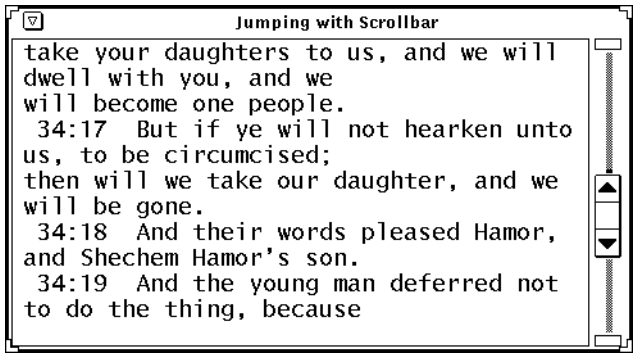


Figure 5-4. Moving top to here: after

5.4.2 Splitting with the scrollbar

A unique feature of the OPEN LOOK scrollbar is that it can be used to "split" the view in most applications. You often want to see two (or even more) parts of a file, and possibly copy text from one place to another. In the next section we'll see how to copy text. Here we'll learn how to get multiple views of the *cmdtool* history. You can split views either by using the *Scrollbar* menu item *Split Views* or by using the cable anchors. Let's say we have a *cmdtool* window that we've been typing commands in for some time; it may look like Figure 5-5:

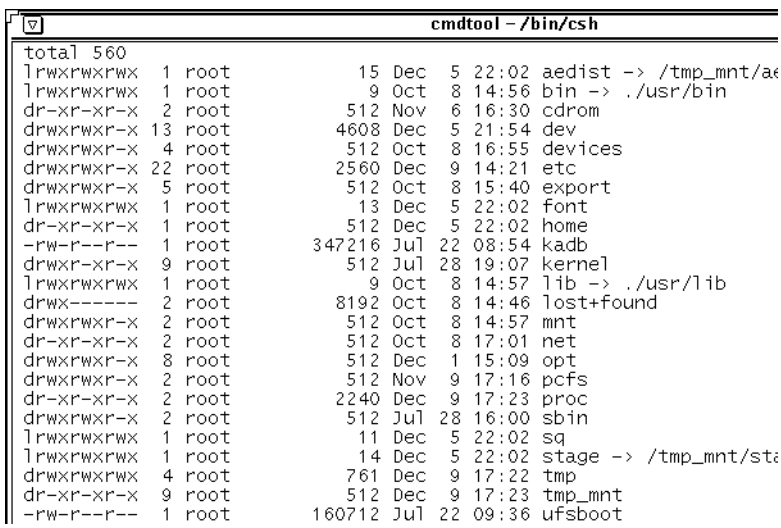


Figure 5-5. Cmdtool window before splitting

To split the window into two views, move the pointer into the scrollbar and press the MENU button, and you will see the menu in Figure 5-6.When you release the MENU button on *Split Views*, you will have two views, each with its own scrollbar, as shown in Figure 5-7

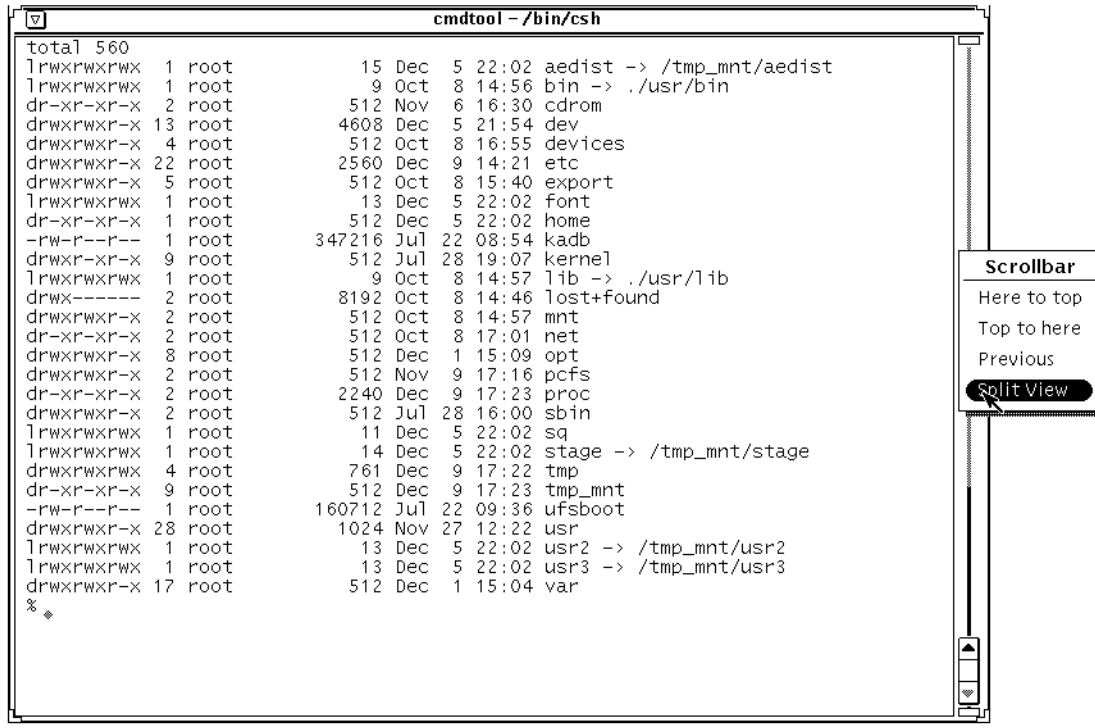


Figure 5-6. Cmdtool window: scrollbar splitting

Notice their initial positions; the top one is left where it was, and the new one begins at the top of the file. Furthermore, their relative sizes reflect where you pulled down the menu; to get two views of equal size, click the MENU button near the middle of the scrollbar. As an interesting experiment, to prove to yourself that these are but multiple views on a single shell history, move both scrollbar elevators to their bottom anchors, and type a new command such as *date* and run it. You should see the command, and its output, appear simultaneously in both views, as shown in Figure 5-8.

You can also split a view by dragging the cable anchor. For example, click and hold SELECT on the top cable anchor, and drag it down: notice that a thin horizontal line follows it, to show the position more clearly. Drag the anchor and line to the place where you want the split to be, and let go. Now you have two views into the data.

If you decide you no longer need multiple views, you can join them again. When you have split views, the scrollbar menu gains an additional last item, *Join Views*; if you release here, the views will be joined. You can also join views by dragging one cable anchor of a view on top of the other cable anchor in that view; the view will be joined with the other view.

Finally, you can have as many split views as you need. Two, three or four split views are not uncommon. An application that uses horizontal scrollbars, such as for viewing a large bitmap or for viewing long lines of text, can allow you to split the view into several vertical views as well. It is up to the application program whether you can split a given scrollbar or

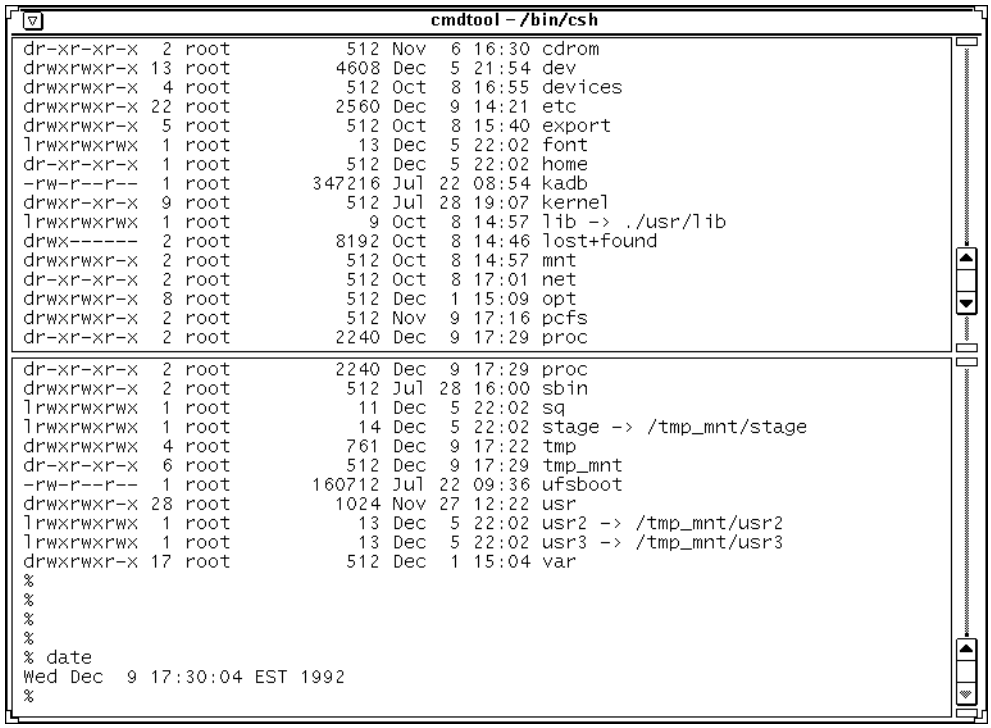


Figure 5-7. Cmdtool window: after splitting, two views

not. If a given scrollbar isn't splittable, it won't have *Split View* in the scrollbar menu. Everything that we have said here about scrollbars applies to scrollbars used in *any* OPEN LOOK application, not just *cmdtool*.

5.5 Copying and Pasting Text Selections

A powerful capability of most OPEN LOOK text application programs is the ability to copy and paste portions of text. You can select a few letters, a word, several words, a whole line, or even several lines. Once selected, text can be copied into an internal buffer and then *pasted* or copied into the input once or as often as you wish. This selection mechanism is similar to, but not identical with, that provided by the MIT X versions of *xterm* and related programs. First we'll see how OPEN LOOK does it, then we'll look at cutting and pasting between OPEN LOOK applications and MIT-style programs.

5.5.1 Selecting Text to Copy

To select some text for copying, you must first move the pointer to the start of the text. Then, you press SELECT, and can either drag the pointer ("wipe") across the text or mark your selection using the pointer buttons.

To wipe across the text, just hold the SELECT key down and move the pointer. As you go, you'll see that the text is highlighted by displaying it in reverse video, as in Figure 5-9†

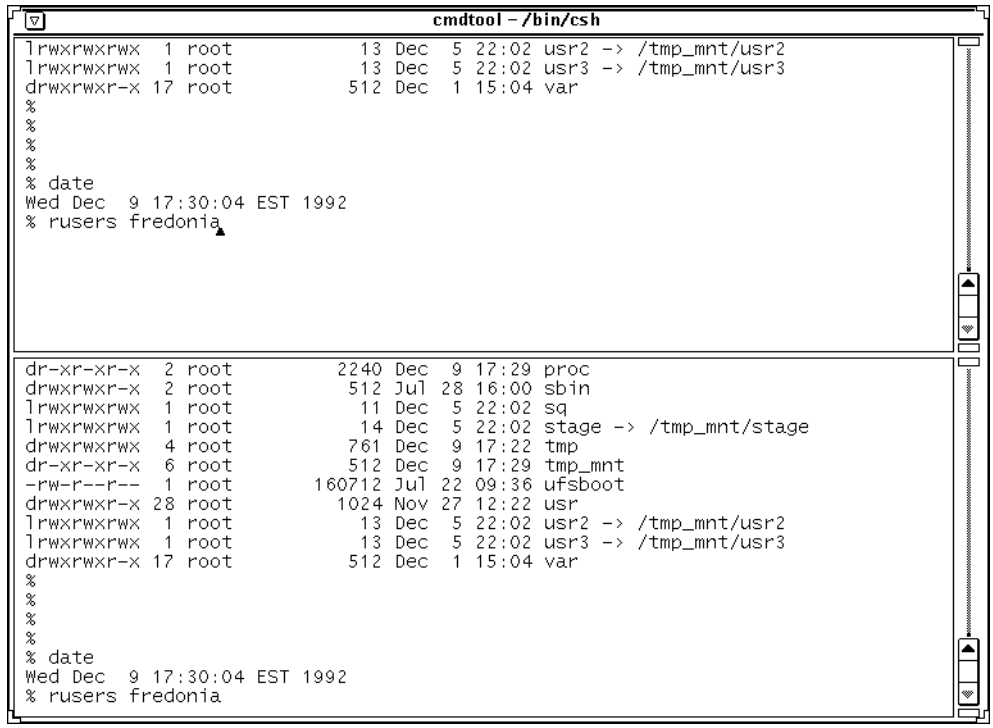


Figure 5-8. Both views reflect a single reality

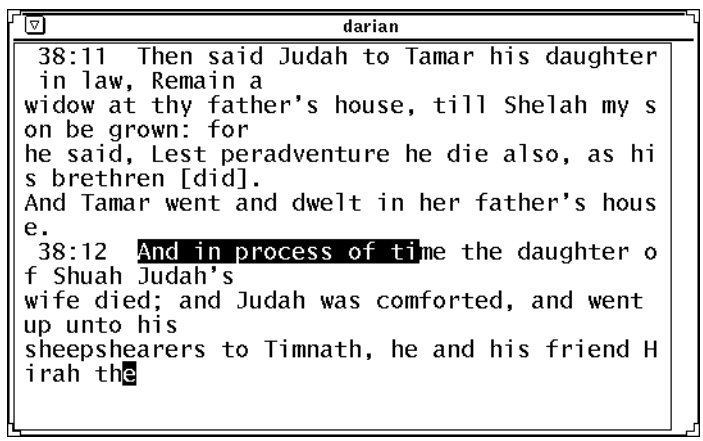


Figure 5-9. Selecting Text by "sweeping"

† Remember that this reverse video is different from the single-character *vi* cursor, if you are using that editor. *Vi*'s cursor shows where it thinks the action is; the selection highlighting shows what text has been selected for later re-use.

You can let go at any time, and that is where the selection will stop. As usual, you can lengthen or shorten the selection with the ADJUST (middle button). With a bit of practice, you can lengthen or shorten it by as little as one character at a time.

Another method of making a selection is by multiple clicking. Clicking and releasing the SELECT pointer button once sets the point at which the selection will start, as before. But clicking and releasing it again very quickly (a *double click*) will cause the selection to include the entire word around the selection starting point:

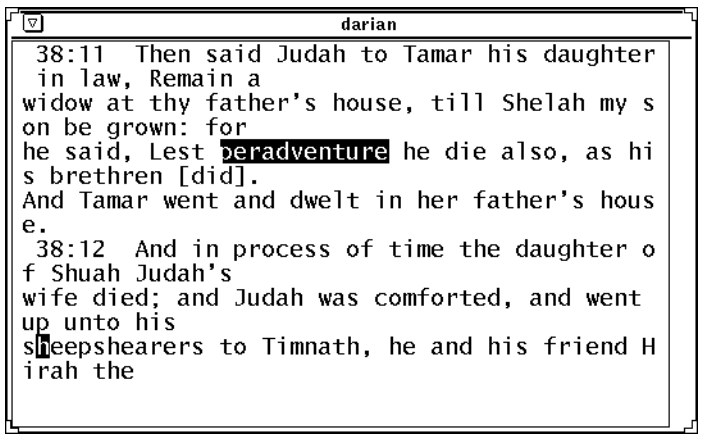


Figure 5-10. Selecting a word by double-clicking

You can also exercise your fingers in a blaze of clicking: three quick clicks in a row, or a *triple click*", will select the entire line on which the pointer is. And if you are really quick, you can *quadruple click* or type four quick clicks; this selects *all* the text that is visible in the window.†

Now you have some text selected, and you can act on it.

5.5.2 Copying or Cutting the text

Once you have some text selected, you can move or copy the selected text into an internal text buffer. If you move it, the original copy is removed from the screen and put into the selection buffer, and this is called a *cut* operation. If you copy it, the original text is left on the screen as well as being put into the selection buffer; this is called a *copy* operation. The *cut* operation is only available while scrolling is on, that is, in *cmdtool* mode but not running a full-screen editor or other application. Again, note that in a *cmdtool* window, this cutting and copying is independent of any cut-and-paste logic in the program you are running, such as *vi*'s Yank and Paste commands or *emacs*' kill-text and yank operations. Also, cutting (as opposed to copying) is automatically disabled when running screen editors *vi*, *emacs*, etc.) or any other program that takes over control of the emulated terminal's screen.

† The Properties editor *props* lets you configure a maximum interval at which a pair of mouse clicks is treated as a double-click.

To copy text that has been selected, you can use the application menu item *Copy*. In *cmdtool* mode this is found under the *Edit* item; in *shelltool* mode, it is on the main menu. Because these are common operations, however, there are normally keyboard shortcuts. On a Sun keyboard, for example, the L6 key is a shortcut for the *copy* operation, L8 is short for *paste*, and L10 is short for *cut.*

5.5.3 Replacing the text

As a shortcut, you can *replace* selected text from the keyboard just by typing into the window or text field that contains the selected text. This in effect combines a *cut* operation with an insert, all in one move.

5.5.4 Pasting Text Selections

A common operation is to copy a section of text from one *cmdtool* window into another, or into a different place in the same window. If you wish to re-use a chunk of text that has been copied (or cut), use the *Paste* operation to insert it. If you want to insert it into text that you are editing with *vi* or *emacs*, for example, move that editor's cursor to the place you want it inserted, make sure the editor is ready to receive it (in the case of *vi*, for example, you must be in insert mode), and choose *Paste* from the application menu; it's found in the same place as *Copy* was found above. The keyboard shortcut on a Sun-3 keyboard is L8. For example, if we wish to paste a longer selection of the Biblical text from above into this *vi* window:

Figure 5-11. Vi window before pasting

we need only make sure that *vi*'s cursor is at the right place, and type (for example) an "o" command to Open a new line and enter Insert mode, and we can paste the text. We can either pull down the window menu and release on Paste. This is shown in Figure 5-12

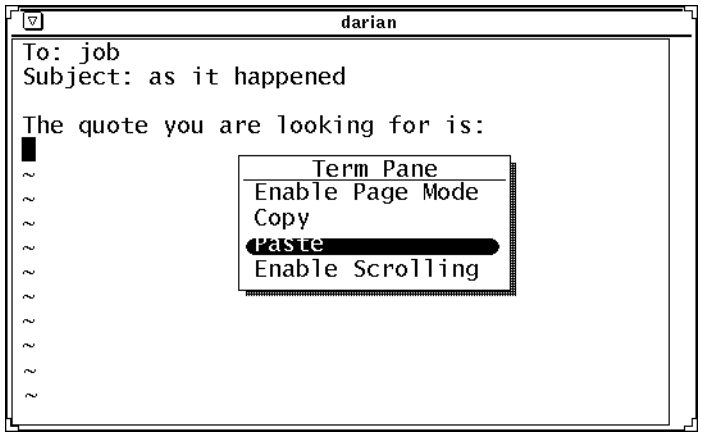


Figure 5-12. window during pasting

Or we could type the "o" command then hit the Paste key (L8). In any event, the text will be pasted in, as shown in Figure 5-13. You will wish to disable the *showmatch* and *autoindent* options in *vi* if you are using them, to avoid unwanted interactions.

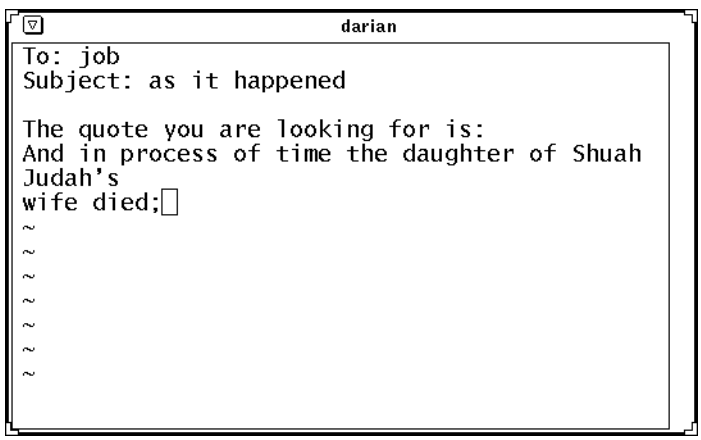


Figure 5-13. Vi window after pasting

If you wish to run the selected text as (part of) a command, be sure you are at a shell prompt in your *cmdtool* window, and paste the text in. For example, here we are about to copy and paste a job name that showed up in the output of a *uustat* command, to re-use it as the argument of another. The syntax

```
uustat -k jobname
```

is used to kill a previously submitted *uucp* (UNIX-to-UNIX copy program) job. This is a good example, since the names that *uustat* generates on this version of *uucp* tend to contain case changes and letters and numbers mixed together, so they are tedious to retype. After

typing the part of the command that doesn't include the jobname (i.e., the "uustat -k"), we select the one word of text we want ("sqN7e04") just by double-clicking on it:

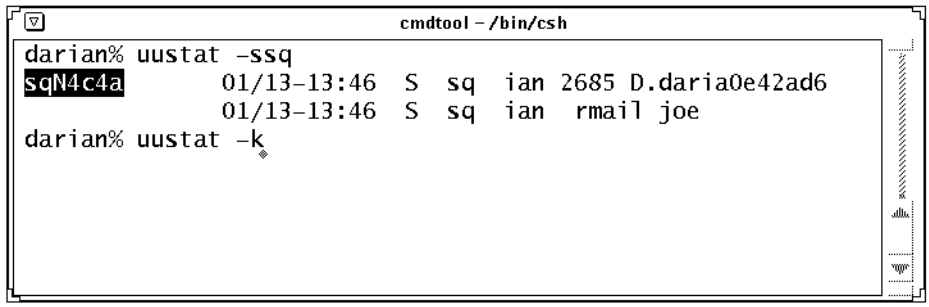


Figure 5-14. Re-using part of a shell command: before

Then you have two choices:

- press the keyboard keys for COPY (L6) and PASTE (L8) to insert the text.
- Click and hold SELECT pointer button, and drag a copy of the selection to the insertion point (as shown in Figure 5-15), then release the pointer button

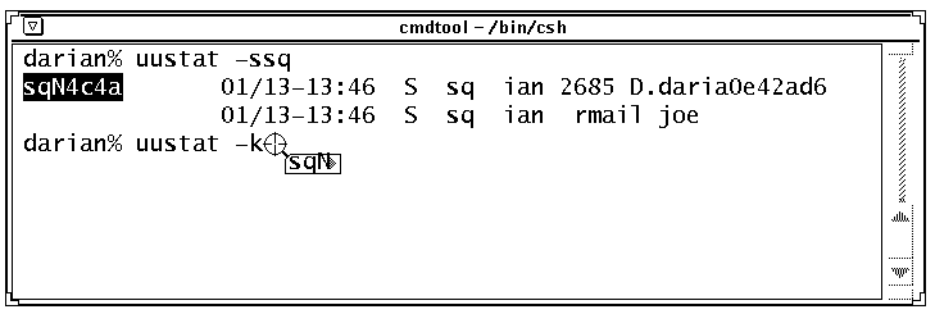


Figure 5-15. Dragging the text with the cursor

When you've copied the text, you may need to press the RETURN key to run the command. The resulting screen should look like Figure 5-16.

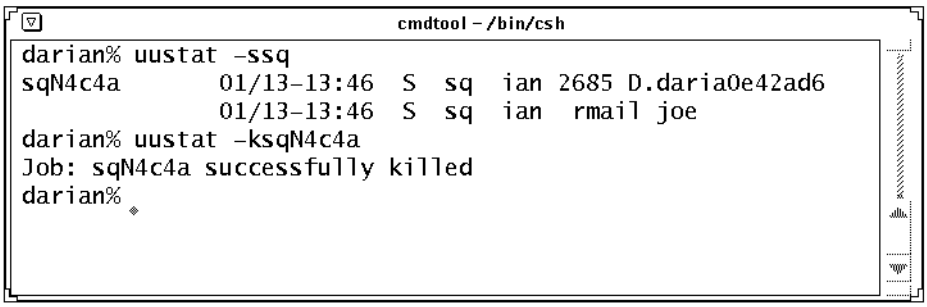


Figure 5-16. Re-using part of a shell command: after

If you have picked up some text from earlier in the history log by using the scrollbar to move backwards, you have to get back to the insertion point at the very end of the history (the current or active line). You can just click SELECT on the bottom scrollbar anchor to move the scroll to the bottom line (this does not de-select any highlighted text you may have), then click on the last line (after the shell prompt). This is a common operation, so Control-Return (press RETURN while holding the Control key) is a shortcut for this, as is the End (R13) key on a Sun Type 4 keyboard. (By contrast, *xterm* moves to the end of the last line automatically when you type anything, even a space; this is a bit faster, but is a consequence of *xterm*'s read-only history log.)

If you want to copy text from one OPEN LOOK window to another, here is the sequence of steps:

- In the source window, select the text;
- Click and hold the SELECT pointer button, and drag the text to where you want it. Note that as in Figure 5-15, the cursor becomes a "gunsight" when you are over a drop target.
- Release the pointer button, and the selection will be copied.

There are some occasions when you cannot use this mechanism. A common case is the *rlogin* and *script* programs, which put the terminal into so-called *raw mode*, in which drag-and-drop does not work (more generally, command-line editing with the mouse does not work). In this case, use the more explicitly copy-and-paste method:

- In the source window, select the text;
- Copy the text into the paste buffer (use *Copy* menu item or *Copy* keyboard button, L6);
- Move the pointer to the new window, and if using click-to-type focus, click SELECT in the target window;
- If using an editor in the target window, make sure the editor is in insert mode at the appropriate point in your file. Alternately, if you want to paste the text into an OPEN LOOK Text Field, click SELECT on that text field to move the insertion caret there.
- Paste the text (use *Paste* menu item or *Paste* keyboard button, L10).

Finally, there is one even shorter method:

- Click SELECT to get an insertion caret where you want the text, if necessary
- Press *and hold* the Paste key (L8)
- Select the text you want copied using the SELECT pointer button. It should be underlined, rather than highlighted in reverse video.
- Let go of the Paste button, and the text will be copied.

To *move* text from one window to another rather than *copying* it, you would follow the same steps for any of these methods except use *Cut* instead of *Copy*.

It sounds like more work than it is. Try it a few times and you'll see how easy it is. For anything with tricky typing, or anything longer than a word or two, it is quite a bit faster (not to mention more accurate) to copy it rather than to re-type it. Once text is in the computer correctly, *let the computer do the work*.

5.5.5 Editing and Saving the History Log

The *cmdtool* program has the ability to save its History Log into a file, using the main menu's *History* menu. This can be used, for example, to document how a program actually behaves, for purposes of writing technical documentation or for filing a bug report. You can edit the *cmdtool*'s history log prior to saving it. Editing can be used, for example, to remove errors that you made, such as running some command with an option missing, without having to flip into a text editor.

5.5.5.1 Clearing the log - a clean start

If you have been using your *cmdtool* session for a while before starting to make a record of some program's behavior, you will probably want to start with a clean slate. Pull down the main menu, then pull right on *History* and further right onto *Clear Log*. When you release this item, the history log will be cleared out, and you can start typing your example.

5.5.5.2 Other Editing

There are many other editing options, described in the following section. These can be used in many places where OPEN LOOK applications allow editing of text fields or strings.

5.5.5.3 Saving

Once you have the log in the form you want it, you can save it to disk using the *History* menu item *Store log as new file*. Just release on this and you will be prompted for the name of a file to save the history log into. Once you type the name and click on the *Save* button, the history log will be written into a file.

5.6 Editing Text in OPEN LOOK Applications

OPEN LOOK specifies the methods used for editing from the mouse and keyboard. This is not to prevent use of more sophisticated editors, such as *vi* and *emacs*, but only to provide a standard set of editing functions that can be used anywhere in the X and OPEN LOOK environments that text must be edited from the mouse and keyboard. This discussion of

editing the history log applies to any OPEN LOOK application that uses text or text fields, such as the OPEN LOOK-specific text editor program, *textedit*, as well as to *cmdtool*'s history log. The OPEN LOOK text editing facilities give you a comprehensive text editor, including cut/copy and paste, search and replace, and the ability to filter text through a series of external programs.

Cutting text out has been described above: select the text using the mouse, and cut it using either the keyboard shortcut (L10) or the *Cut* main menu item.

You can insert text at any point. Position the pointer where you want to begin inserting, and click SELECT. You will see a small insertion caret, and can begin typing at once. Continue typing until you have said your piece. If you make a mistake, you can correct it with the backspace key (some applications also let you use the DELETE key). Most other keys, including control characters, will be inserted directly.

The main menu, called *Term Pane* in *cmdtool* and *Text Pane* in *textedit*, has an *Edit* item that invokes a series of editing functions, most of them familiar. The bottom three are *Copy*, *Paste*, and *Cut*, which have been described earlier in this chapter. The others are *Again* which repeats the previous editing change, and *Undo*. *Undo* can undo either the single most recent editing change, or the entire set of editing changes. The order of these change operations, *Again*, *Undo*, *Copy*, *Paste*, and *Cut* may seem arbitrary, but it is the same order as the keyboard shortcuts for these actions on the Sun-3 keyboard: Again is L2, Undo is L4, Copy is L6, Paste is L8 and Cut is L10.

The next item in *cmdtool*'s *Term Pane* and *textedit*'s *Text Pane* is *Find*, which lets you locate and optionally replace text. It leads to a menu of four items: *Find & Replace...*, *Find Selection*, *Find Marked Text*, and *Replace Field*. The first, *Find & Replace...*, pops up a window with two text fields, one for the text to be found and the other for the text to be replaced. There are several push buttons on this pop-up window:

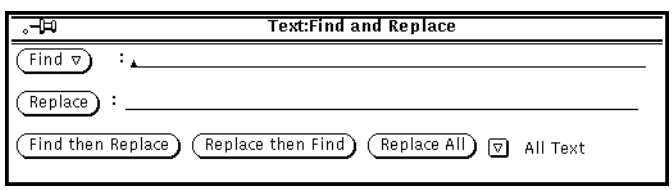


Figure 5-17. Find and Replace Pop-up Window

The *Find* button lets you search forward or backwards after you have entered your text. If you wish to replace the found text, you can select *Replace then Find* to do the replacement and find the next occurrence. Or you can select *Replace All* which will do the replacement in the whole file. The abbreviated menu button at the right lets you select between *All Text* (replace throughout the entire document) or *Here to end* which will only make the change from the current point or found string up to the end of the document. The *Find Selection* item lets you select some text and find it, without having to copy it or retype it in the *Find* text field. The next matching text will be found, if there is one, and left selected, so you can cut or copy it from the new location. This is a common operation, so there is a keyboard shortcut for it. On a Sun keyboard, the L9 (FIND) key will find the selected string if you are in an edit window. Shift-L9 will find backwards. On any keyboard, Meta-F and Meta-

Shift-F will perform the same find operations. If there is no string currently selected, it will try to find the most recently *Cut* or *Copied* string.

Find marked text and *Replace Field* are advanced topics and will not be described here.

The next item in *cmdtool*'s *Term Pane* and *textedit*'s *Text Pane* is *Extras*, a list of odds and ends that runs some external filters on the selected text. The *Format* item runs the *fmt* text alignment program; this is useful for making long lines of text fit, or for joining words into a single line to run as a command. For example, to list (*ls*) the three files highlighted in Figure 5-18,

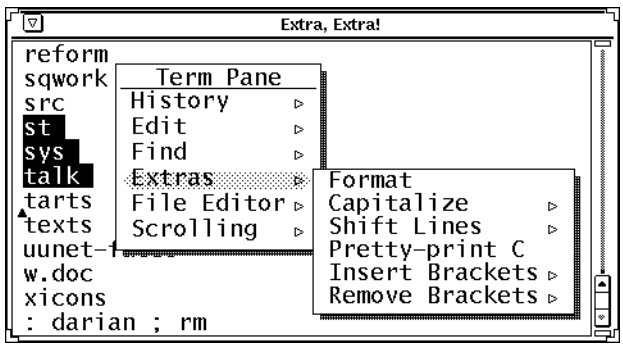


Figure 5-18. Three filenames selected for "formatting"

we select them with the pointer, pull down the *Term Pane* to *Extras*, and release on *Format*. This replaces the three lines shown, with one line:

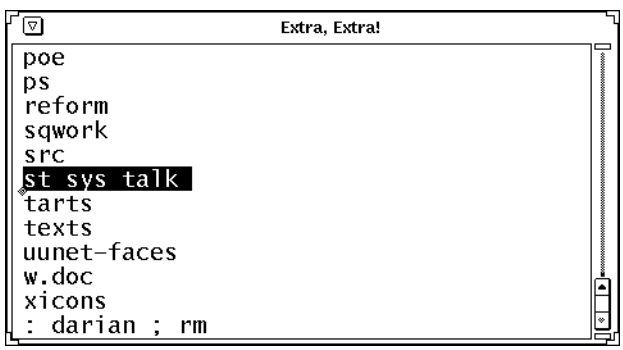


Figure 5-19. Lines joined together

We then highlight this line (either by wiping across it or by triple-clicking), press the COPY key, move to after the *rm* command (either by clicking SELECT to the right of it, or by using Control-RETURN), press the PASTE key, and it's done:

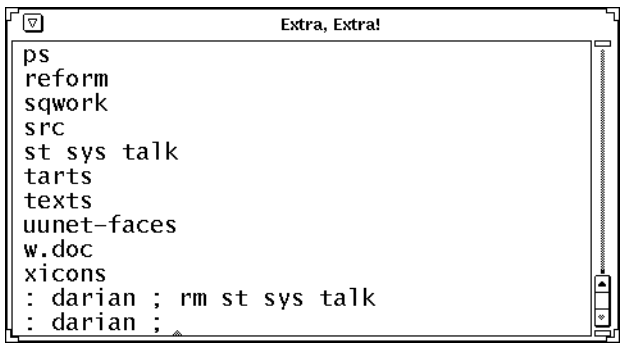


Figure 5-20. The command is done

Admittedly cutting and pasting is not much faster for three short filenames, but for half a dozen long or complicated filenames it can be a significant time-saver.

The remaining items in the *Extras* menu allow you to

- shift the case of text from upper to lower (useful with MS-DOS filenames), lower to upper, or even to capitalize words (useful with people's names);
- shift entire lines to the left or right;
- have C source code reformatted or "pretty-printed", and
- insert and remove various kinds of parentheses or brackets (useful in expressions in most computer programming languages.

These are mostly useful to programmers who already know why these tools are necessary, so they're not explained here. We will show you how to change the functions in this list in **Snuff this dead chapter reference ASAP!**.

The final item in *cmdtool*'s *Term Pane* is *File Editor*. This selection turns the bottom half (or more) of the *cmdtool* screen into a *textedit* window, which behaves just like *textedit*, as described in the next section.

5.7 Editing with Textedit

Textedit is the OpenWindows text editor program. It can be started from the *Workspace* menu, from the OPEN LOOK File Manager, from within a *cmdtool* window, or any place else you need editing abilities. *Textedit* is easier to learn than conventional UNIX text editors like *vi* and *emacs*, but an adept user of one of those editors can get work done more quickly than a user of *textedit*.

All of its editing facilities are the same as those of *cmdtool*, since it contains the same editing program. All you need to learn now is how to change text in the on-screen buffer, and how read and write files plus a few other, less common operations and you'll be a *textedit* user.

5.7.1 The Editing Keys

There are several keys that can be used to edit text in an OPEN LOOK application such as *cmdtool*. These are similar to those in other keyboard editors, most notably to the *emacs* family of editors. But they are not exactly the same as *emacs*, so users of that editor should not get complacent, and users of other editors should not feel disadvantaged.

Before describing the individual keys, we need to explain the notion of *insertion caret* or *insertion point.* There is always one insertion point in the window of an OPEN LOOK application that has any text fields or text windows. This is the point at which most editing operations take place. For example, text that you type is inserted at the caret, while deleting the start of a line deletes from the caret leftward to the start of the line. You can move the insertion point anywhere in the text window just by moving the pointer to where you want the caret, and clicking SELECT. A small caret symbol (♦) will be displayed there.

Several control keys (hold the Control key and type the given letter) have effect on the edit. These are similar to the MIT *xedit* program, however, the list of keys that work here is smaller than with *xedit*. Several keys that work in *xedit* but not here are listed as "No effect" so that you can compare them if you have previously used *xedit*. Table 5-1 lists the control keys and their actions.

Table 5-1. TextEdit Editing Characters

Key	SHIFT reverses?	Function (mnemonic)
Control-A	yes	Move to the beginning of the current line (previous line if repeated).
Control-B	yes	Move backward one character ("backward").
Control-D	no	No effect; inserts a Control-D.
Control-E	yes	Move to the end of the current line.
Control-F	yes	Move forward one character ("forward").
Control-H	yes	No effect; inserts a Control-H.
Backspace	yes	Deletes char to left of cursor
Return	no	Break line at caret.
Control-Return	yes	Moves to end of document
Control-I	n/a	Inserts TAB character.
Control-K	n/a	No effect; inserts itself.

Table 5-1. TextEdit Editing Characters

Key	SHIFT reverses?	Function (mnemonic)
Control-L	n/a	No effect; inserts itself;
Control-N	yes	Move down to the next line. (“next”)
Control-O	n/a	No effect; inserts itself.
Control-P	yes	Move up to the previous line. (“previous”)
Control-U	yes	Delete from the start of line to the insertion point (common *stty* setting)
Control-V	n/a	No effect; inserts itself
Control-W	yes	Delete the word to the left of the caret. (common *stty* setting)
Control-Y	n/a	No effect; inserts itself.
Control-,	no	Word motion to left (under < key).
Control-.	no	Word motion to right (under > key).
Escape key	n/a	No effect; inserts itself.
Delete key	yes	Delete the character to the left of the caret.

5.7.2 The textedit menus

When started from the *Workspace* menu or the command line with no filename argument given, *textedit* looks like this:

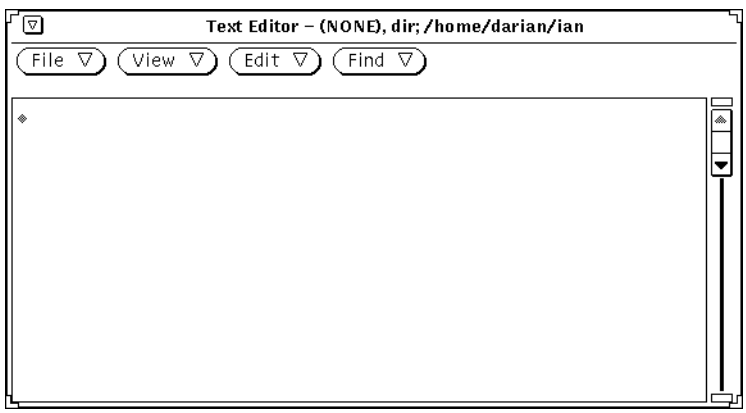


Figure 5-21. Textedit initial screen

The "NONE" in the titlebar simply means that no file has been selected. When enabled within a *cmdtool*, the text editor looks similar except that it only occupies the bottom part of the window; the bottom of the command log is displayed in the top window, like so:

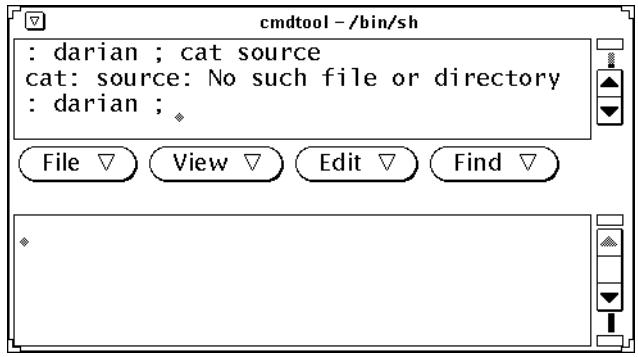


Figure 5-22. TextEdit started from within cmdtool

In either case, you are presented with an empty document. If started with a filename on the command line, of cours4e, *textedit* starts up with the first screenful of text displayed. To create a new document, just begin typing. Or you can load an existing document using the *File* menu. Here is an example of using the editor within *cmdtool* to create a new, short file:

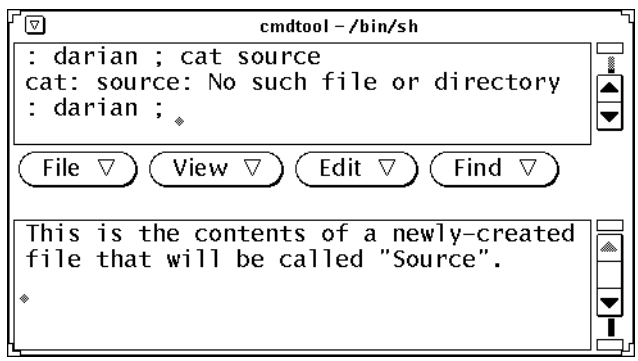


Figure 5-23. Creating a short file with textedit.

Having created the two lines of text, you would save it as a new file, as described below using the *Edit* Hint: you can just click SELECT on the *File* button and the program will do the right thing: save an existing file, or ask you for a name to save a new file under.

The menu bar has the conventional *File*, *View*, *Edit* and *Find* pulldown menus. Each of these pulls down a menu. In addition, these are the same names and in the same order as the first four controls in the *Window Menu*, which also has *Extras.* But *Edit*, *Find*, and *Extras* are the same in *textedit* as in *cmdtool* above, so there is nothing new to say about them. Instead, let us discuss *File* and *View.*

5.7.2.1 The File menu

The *File...* menu, whether accessed from the menu bar or the window menu, has these choices in it:

Load File...
Save Current File
Store as New File...
Include File...
Empty Document

Load File... is used to load a text file into the editor to start editing it. You get a filename dialog window, either a directory browser or a simple dialog with text fields for the directory and filename, and a *Load File* button. The directory text field is initialized to the current directory, so if the file is in the same directory, just type its name and click the *Load* button. If the file is in some other directory, then you can either edit the directory name and type the filename, or just type the full path in the *Filename* field. In either case, click *Load* to actually load the file. And as a shortcut, hitting RETURN after typing the filename will load it, just as if you had clicked on the button. Many OpenWindows programs use this technique: notice that the *Load* button has a double circle or "default ring" to indicate that it is the default action. If you have popped up this dialog but want to cancel the action, you can pull down the window menu of the popup window and select *Dismiss*, which will cancel the Load operation.

When there is nothing in the edit buffer, the default item on the *File* menu is *Load file...* Once you have started editing, the default item changes to *Save Current File*. Clicking SELECT on a menu button performs the menu's current default action. Also, if you try to load a file when you have unsaved text in the text buffer, you will be prompted to save it before it is cleared.

Save Current File saves the contents of the *textedit* buffer back into the file you edited it from.

Store as New File lets you write the changed file in the edit buffer out to a different file than that from which you edited it. This would also be used in the case where you are creating a new file with *textedit*. This popup dialog behaves the same as the one for *Load File*

The *Include File* control is used to insert another file in the place where the insertion caret is, just as if you had typed the file in anew. It prompts with a standard File popup dialog, as above. When you click on the *Include* button, the file is read into your edit buffer, right where the insertion caret is.

A valuable short form for *Include File* is <Meta>-i (hold Meta key and type "i"). If a filename is selected anywhere on the display, it will be pre-loaded into the *Include* dialog, and you just have to hit return there (or click on the *Include* button) to insert the file. If the selected filename is in the current *textedit* window, it will be *replaced* by the contents of the file.

The *Empty Document* control does what it says: it empties the document, or gives you an empty document. If you have unsaved text in the buffer, you are prompted with a Notice.

A successful *Empty* operation forgets the remembered filename, too, so you don't have to worry about accidentally saving an empty file over top of one you were editing.

5.7.2.2 The View menu

The *View* menu has these controls:

Select Line At Number...
What Line number?
Show Caret at Top
Change Line Wrap

Select Line At Number... simply allows you to jump to a given line by number, and select all the text on that line for copying. If you want to get to line 450 of the file you are editing, for example, select this control and you will pop up a small dialog box into which you can type the number 450. When you press the *Select* button, your view of the file will be advanced to line 450, and the text of line 450 will be selected so that you can Cut or Copy it for later pasting:

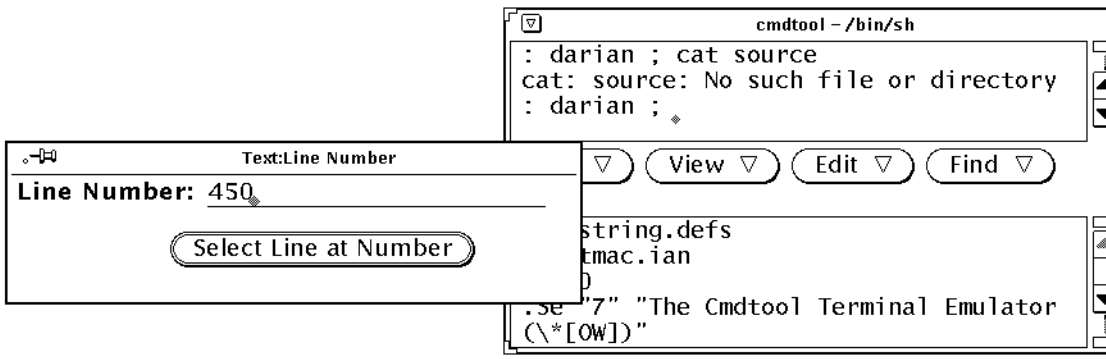


Figure 5-24. Selecting line 450

When we click the *Select* button, line 450 will be highlighted:

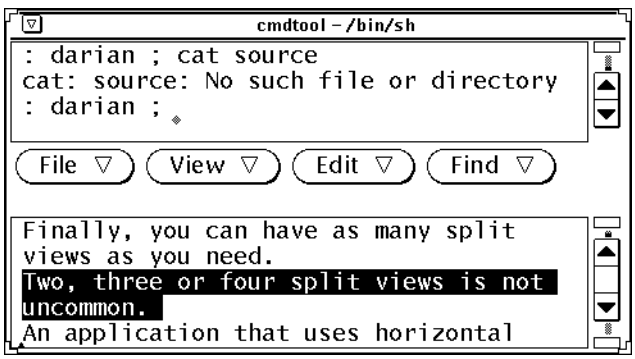


Figure 5-25. Line 450 now Selected

A sort of inverse operation is *What Line number?*, which tells you the line number that the currently-selected text begins on. Let us suppose that we have selected this text from a larger file, and want to see where it is in the file:

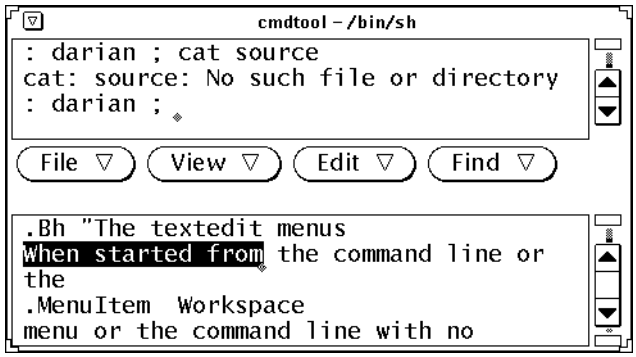


Figure 5-26. What line is this text on?

We pull down the *View* menu and select *What Line Number?* Immediately, this Notice pops up:

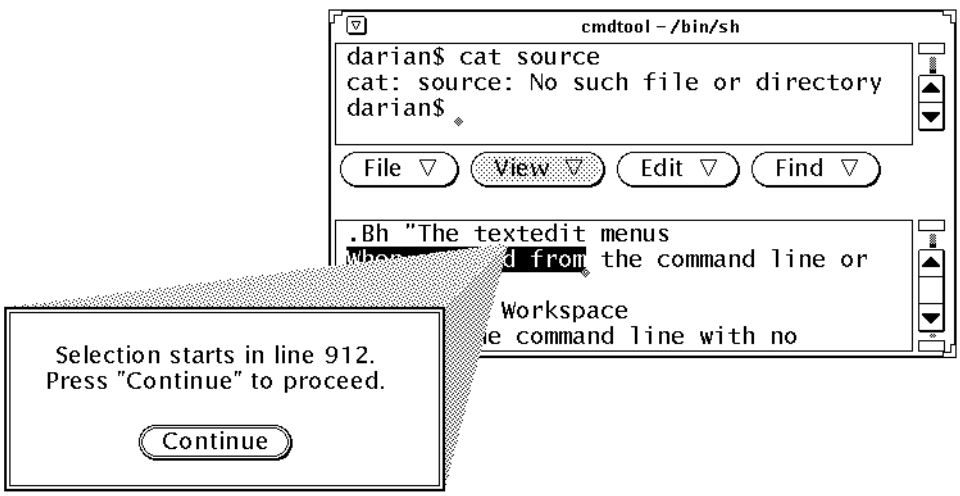


Figure 5-27. It is on line number 912.

and we see that is line number 912 in the file that we are editing.

The menu item *Show Caret at Top* simply moves the text so that the line with the insertion caret is at or near the top of the screen, so you can see the context around it better. You could do this yourself with the scrollbar, so I'm not sure why you'd want it.

The final item, *Change Line Wrap*, lets you control how lines longer than the width of the text window are to be displayed. You can choose from *Wrap at Word*, *Wrap at Character*, or *Clip lines*. The first two are reasonably self-explanatory; text that won't fit on a line is wrapped either at a word boundary or at a character boundary (i.e., right at the right mar-

gin). The last item doesn't actually truncate the lines in the file, but only ignores them for purposes of the display. In fact, none of these options actually changes the text; they only affect the way it is displayed. If you find yourself setting this each time you start an edit window, you may wish to customize it, as described in **Snuff this dead chapter reference ASAP!**.

5.7.2.3 The Edit, Find, and Extras Menus

As mentioned, the *Edit*, *Find*, and *Extras* in *textedit* have the same controls and function in exactly the same fashion as they do when editing a history item in *cmdtool*, so they are not discussed further here.

This completes our discussion of the Text Editing mode of *cmdtool* as well as of *textedit*. We now discuss a few more details about Text Selections, and show you how to start a *cmdtool* window to run one specific program instead of running a shell.

5.8 More About Text Selections

Since the MIT xterm program is so widely used, you may on occasion need to transfer text between cmdtool and xterm, or vice versa. This section discusses the details.

5.8.1 Copying and Pasting between XView and MIT Clients: xcutsel

You can select text in *cmdtool* (or *textedit*, or any OPEN LOOK client that has a text field or a text window), then move the pointer to an MIT *xterm* and click the middle button. Your text will be pasted into the *xterm*. This technique also allows you to copy text into *xedit*, *xclipboard*, and other MIT-style clients.

But to copy from an MIT *xterm* (or *xedit* or *xclipboard*) into an XView OPEN LOOK client, you have to use the following procedure:

- Select the text in the *xterm*, using *xterm* conventions (sweep with Button1 (left), extend if necessary with Button3 (right button));
- Move the pointer into the *xclipboard* main window;
- Paste the text into the *xclipboard* window using the *xterm* conventions (Button2, middle button, pastes text);
- Move the pointer into the OPEN LOOK window;
- Paste the text using the *Paste* button or the *Paste* item from the *Edit* menu.

5.8.2 Saving Multiple Selections: xclipboard

Xclipboard lets you have multiple selections. Its *New* button creates a new selection; once you have created more than one, the *Next* and *Previous* buttons move you among them.

5.8.2.1 Editing Text Saved in xclipboard

It is possible to edit text in an *xclipboard* window, but you should be aware that it uses the *xedit* conventions, which are similar to *textedit*, but not identical. Refer to the discussion of

textedit in *Chapter 5, The cmdtool/shelltool Terminal Emulator,* and of *xedit* in Chapter 8, *Other Standard Clients*, for more details.

5.9 Running a Program in a Temporary cmdtool Window

Normally you start up a *cmdtool* window and keep it running for a long time, typically the duration of your session. Each such *cmdtool* has its own copy of your login shell (more precisely, the program named in the SHELL environment variable) or the shell named in your *.Xresources* file. You can, however, start up an *ad-hoc* copy of *cmdtool* just for the duration of a single command. All you do is put the name of the command (and any arguments to it) *after* any generic XView arguments:

```
% cmdtool [generic-arguments] command [command-arguments]
```

For example, if you want to run *more* on a file named *temp* in a window that will disappear when you quit (or reach end of file), you can do so as follows:

```
% cmdtool more temp
```

The generic arguments (such as -display) must appear before the command, and arguments to the command must be last. For example, to run the above command with the output appearing on the second screen of a two-headed display,

```
% cmdtool -display unix:0.1 more temp
```

Be aware, though, that if the command terminates without reading from the terminal (as would be the case if file *temp* were shorter than one screen full of text), there is a good chance that it will terminate before the window is fully displayed, in which case you will not see any output. One way around this is to use a *sleep* command, as in

```
cmdtool more temp \; sleep 600
```

which will sleep for up to ten minutes (600 seconds) after the *more*. You can interrupt this by a Control-C (or your interrupt character) in the temporary window. Another way is to read from the window's standard input after the command terminates:

```
cmdtool cat myfile \; echo Press Return \; read junk
```

This will wait until you press return before terminating the window.

5.10 Cmdtool as a Console Window

In the olden days of UNIX, there was a special printing terminal set aside for the console operator, and it had the special name */dev/console*. This file is used by many programs to log messages on, and the file still exists in modern UNIX, even though the hardcopy terminal is gone from workstations. On most modern UNIX systems, */dev/console* is a pseudo-device that causes the log messages (actually anything written to */dev/console*) to appear "somewhere else". That "somewhere" may be on a printer, in a file, or on another terminal. In the case of workstations with graphic screens, it is usually the graphic screen, so that the workstation user will see any "important" messages that are sent to /dev/console. But in the

case of a windowing system such as X, it is not a good idea to have programs writing directly to the screen, as this would appear over top of various windows, and jumble up the screen contents (see Figure 5-28).

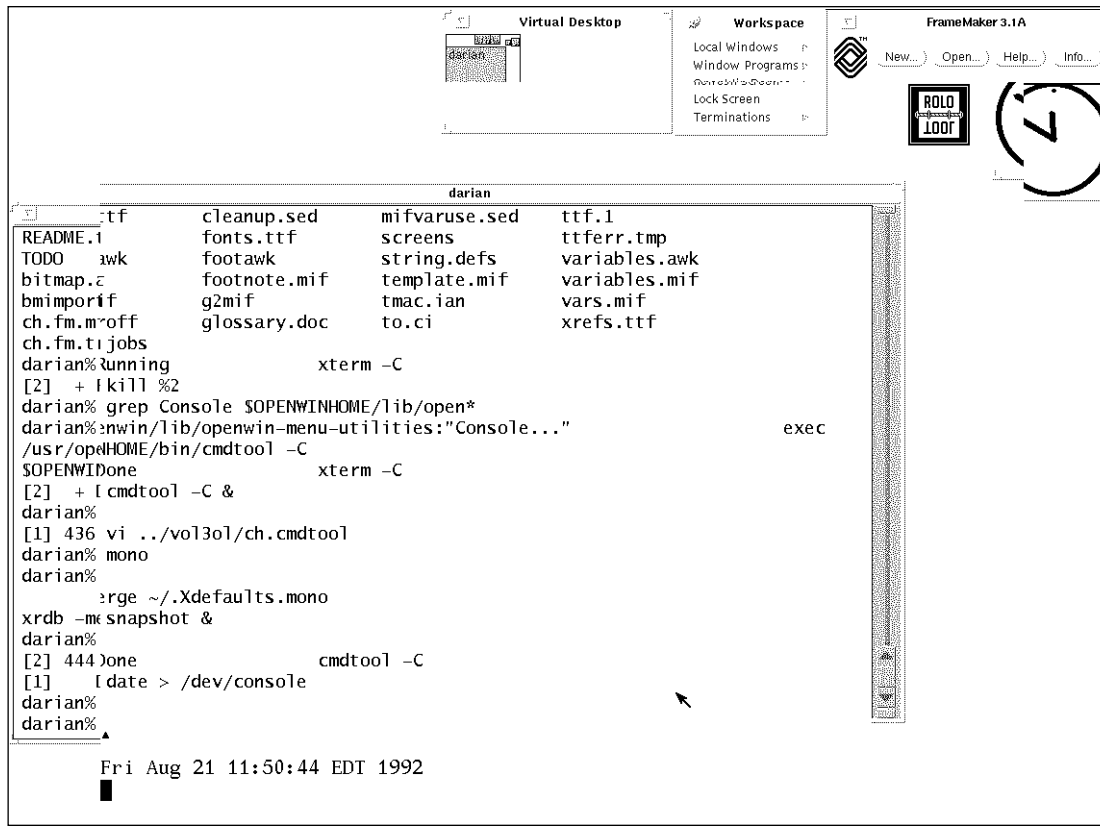


Figure 5-28. A screen with console messages over windows

To solve this problem temporarily, you must *refresh* or *repaint* the screen, that is, repaint the root window and then have each client program repaint its window(s). On OpenWindows this is available in the default menu under *Utilities* on the workspace menu, where it is the default item so you need only click SELEECT on Utilities in the root menu. With other window managers your mileage may vary.

It is better to *prevent* this problem, which you normally do by making a terminal emulator "become the console". The *cmdtool* program (and other terminal emulators such as *xterm*) accept a -C option to make them the console. This is normally specified for the first or last terminal emulator that your session scripts start up, and it is often a small (6 lines by 80 characters) window, to allow you to see messages but not use up much screen space. For example, the default OpenWindows start-up script (*.openwin-init*) has this entry:

```
$OPENWINHOME/bin/cmdtool -Wp 0 0 -Ws 590 77 -C
```

The -Wp and -Ws are standard command line options for position (upper left corner) and size (small) of the window. The -C makes this instance of *cmdtool* become the system console window.

A special-purpose program, *contool*, is available as contributed software. *Contool* is optimized for use as a console; you cannot type commands into it, but it allows you considerable flexibility in filtering out the more common but uninteresting messages that may appear on your console. It can arrange to write some or all messages into a log file, and can open (de-iconify) itself automatically when certain messages appear. See the reference manual in Part Three of this guide for details on this useful (con)tool.

5.11 Cmdtool/Shelltool/Textedit Menus Reference

The chart in Figure 5-29 shows the menus available, starting in *shelltool* mode. For example, turning *Scrolling on* sets the mode to *cmdtool*, where the Term pane menu lists History, Edit, etc. This is the same menu you get if you start off in *cmdtool* mode. Similarly, setting *File Editor Enable* turns on *TextEdit* mode, wherein the menus are the same as if you had started off by running a *textedit* window.

```
ShellTool's Term Pane
Enable Page Mode
Copy
Paste
Enable Scrolling --> CmdTool Mode
        Cmdtool's Term Pane
        History -> Mode -> Editable/Read-Only
            -> Store log as new file...
            -> Clear log
        Edit -> Again
            -> Undo -> Undo Last Edit
                    -> Undo All Edits
            -> Copy
            -> Paste
            -> Cut
        Find-> Find & Replace...
            -> Find Selection-> Forward
                        -> Backwards
            -> Find marked text ...
            -> Replace Field-> Expand/Next/Previous
        Extras-> Format
            -> Capitalize ->
            -> Shift Lines ->
            -> Pretty Print C
            -> Insert Brackets ->
            -> Remove Brackets ->
        File Editor --> Enable
                    Textedit's Text Pane
                    File-> Load File
                        -> Save Current File...
                        -> Store as New File...
                        -> Include File
                        -> Empty document
                    View-> Select Line at #...
                        -> What line #?
                        -> Show caret at top
                        -> Change Line Wrap ->
                    Edit
```

```
                    Find (same as Find above)
                    Extras (same as Extras above)
```

Figure 5-29. Shelltool/Cmdtool/Textedit Menus Reference

5.12 Other Terminal Emulator Programs

Now we've discussed the *cmdtool* terminal emulator in detail. There are many other terminal emulators that you should at least be aware of.

5.12.1 Xterm

Xterm, the MIT standard terminal emulator, is widely used in some parts of the X Window System community. It is discussed in Appendix A, *The xterm/olterm Terminal Emulator*. Since *xterm* is so widely used, a brief comparison is in order: please refer to Table 5-2.

Table 5-2. Shelltool/Xterm Comparison

Cmdtool/Shelltool	Xterm
Cmdtool has the advantage that it complies fully with the OL specification.	Widely used.
Cmdtool works with the OPEN LOOK cut and paste, drag and drop, and other mechanisms.	Xterm has a much simpler copy-and-paste mechanism: you select with button1, extend with button3, and paste the selection with button2, instead of with function keys or a menu.
Cmdtool has a better history mechanism, letting you revise the shell history, save it into a file, etc.	*Xterm* has a limited history: you can select from any previous line or lines, but not change them.
Because of its history mechanism, earlier versions of *cmdtool* had problems with command interpreters that play games with *stty* settings, such as *csh* with "set completion" on, *tcsh*, *bash*, *ksh* with "set -o vi/emacs", etc. These are fixed now.	*Xterm* is faster. It passes interrupt characters (^C, ^Z) much more quickly, and it uses less CPU time in normal operation.
The version of *cmdtool* in OpenWindows 3.0 has problems when scrolling text while a window partly overlaps it; some of the text does not get redisplayed. This will probably be fixed soon, possibly by the time you read this.	Not a problem.
Geometry size in pixels (700x500)	Geometry size in characters (80x24)

5.12.2 Others

jet is an experimental terminal emulator that is distributed in both source code and binary forma as part of OpenWindows 3.0. A related program is *jed*, the jet-like editor. You could say that *jed* is to *jet* as *textedit* is to *cmdtoo*l; the terminal emulators feature simple editing, and the editors work only on text file but have more functionality.

Some commercial X packages include their own terminal emulators. Examples include DECterm in Digital Equipment's DECWindows and AIXterm in IBM's AIX version of UNIX. These are discussed in the appropriate vendor documentation.

Now let's turn our attention to some other X clients. First we'll discuss some additional features of the OPEN LOOK Window Manager. The next two chapters deal with useful tools; they cover the OpenWindows DeskSet tools and the MIT "standard" clients that should be in any release of The X Window System. This is followed by a chapter on Graphics Clients that mentions MIT clients, OPEN LOOK clients, and some free software clients.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 6

More about the OPEN LOOK Window Manager

The OPEN LOOK Window Manager, *olwm*, is one of many window managers available for X11. However, if you are to make full use of OPEN LOOK, you are really committed to using *olwm*, for it is the only Window Manager that implements all that OPEN LOOK requires of a window manager. Both Sun and AT&T provide their own versions of *olwm*; in this chapter we discuss the common functions that they provide for a variety of window management functions. In Chapter 2, *Working in the OPEN LOOK Environment*, we described the OPEN LOOK-specified procedures for selecting the input focus window, raising windows, moving and resizing them, and opening/closing them using the pointer. Here we will discuss a number of additional features of the OPEN LOOK Window Manager. You can use *olwm* to:

- Create additional terminal windows.
- Move windows to a new location on the screen.
- Change the size of windows.
- Lower windows (move them to the back).
- Refresh a window, or the entire screen.
- Close (iconify) or remove (delete) windows.

The OPEN LOOK Window Manager lets you manage your windows in several ways:

- You can use the “window frame” (including the titlebar) and various features on it: the window mark (iconify or “minimize” button);
- You can use the *Window Menu* available from the window’s frame;
- You can use the *Workspace Menu* (which most X11 documentation calls the *Root Menu);*
- Or you can use keyboard keys as shortcuts for many of these operations.

In this chapter, we'll review some basics about focusing input to a window or icon, and consider some other window management functions that can be performed using the window's frame, the *Window Menu*, and the *Workspace Menu*.

We'll see that some window management functions can be done from the keyboard, using specially-assigned keys. These keyboard shortcuts are often called "accelerators" because they can speed up your work. Keyboard accelerators usually involve special keys that X11 calls *modifier keys*. So we'll have to spend a bit of time looking at what X considers "special."

The *olwm* program is moderately flexible – less so than some other window managers, but with the intent of keeping OPEN LOOK intact as a standard – so you can configure it in certain ways. You can change the focus policy from the default click-to-type to a more convenient point-to-type (keyboard focus was described in more detail in Chapter 1). You can also change the *Workspace Menu*; in Sun's version of *olwm* you can replace the entire menu, organizing it as you wish, while in AT&T's version you can only add entries to the *Programs* list in the menu. Chapter 13, *Customizing olwm*, is devoted to this topic.

An even more flexible OPEN LOOK Window Manager is *olvwm*, which is upwards compatible with the OpenWindows version of *olwm* (from which it is derived). It is not a supported product, but the source for it can be obtained by anonymous *ftp* from Uunet and other sites. It has a "virtual desktop" facility that gives the appearance of multiple screens on a single display—as many screens as you want. See Section 6.5, "The Virtual Desktop (Virtual Edges)," for details on *olvwm*.

This chapter is aimed at the user of the OPEN LOOK Window Manager program from either Sun's OpenWindows 3.0 or AT&T's OPEN LOOK GUI 4.0. Use of other versions of the OPEN LOOK Window Manager, or of locally-customized versions, will be similar, but you should expect a few small differences as you go through the chapter.

6.1 Using Special Keys

Undoubtedly you know how to use a keyboard. However, X interprets certain keys somewhat differently than the labels on the keys would indicate.

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

Three of these modifier keys should be familiar to any user of a standard ASCII terminal or a personal computer—Shift, Caps Lock, and Control. However, many workstations have additional modifier keys. A PC has an "Alt" key, a Macintosh™ has a "clover" ("command") key, a Sony workstation has keys named "Nfer" and "Xfer," and most Sun Workstations have three additional modifier keys, labeled "Alt" or "Alternate", and either a diamond character (◊) or "Left" and "Right.

Because X clients are designed to run on many different workstations, with different keyboards, it is difficult to assign functions to special keys on the keyboard. A software developer can't be sure that any given key will exist on every user's keyboard!

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

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

We'll talk at length about this subject in Chapter 14, *Customization Clients*, but we wanted to warn you here. When we talk later in this chapter about pressing the "Meta" key, you should be aware that there is not likely to be a physical key on the keyboard with that name. For example, on Sun workstations with the normal Type 4 keyboard, the key is labelled with a diamond character, on another workstation, the Meta key might be labeled "Alt" and, on another, "Left." And as we'll show in Chapter 14, you can choose any key you want to act as the Meta key.

Unfortunately, if you or your system administrator has been busy modifying key assignments, there is no easy way to find out which key on your keyboard has been assigned to be the Meta key. When you need to know, please turn to the discussion of key mapping in Chapter 14, *Customization Clients*, to learn more.

6.2 Input Focus and the Window Manager

Input Focus is, as described in Chapter 1, *An Introduction to OPEN LOOK and the X Window System*, the direction of keyboard input into a given window by use of the pointing device (mouse). The management of the input focus is controlled by the Window Manager program in any version of The X Window System. The OPEN LOOK Window Manager provides both common focus policies, pointer-focus and click-to-focus, so you can use the system as it suits you. In click-to-focus, you move the pointer into the titlebar of the window you want to focus input on, and click the SELECT (left) button. In pointer-driven or click-to-focus mode, the focus is directed to a window just by moving the pointer into that window. Both focus styles have their uses.

6.2.1 Focusing Input on an Icon

When a window is iconified, or closed to a small icon, it normally does not receive input. But you can still direct the focus to it. For example, if you iconify a terminal emulator that is a shell window, and then move the focus into the icon and type commands, nothing will happen; the keystrokes will be lost. However, there is one exception: the OPEN LOOK Window Manager allows certain keyboard shortcuts - such as opening the window - to be typed to an iconified window. And, in *olwm*, you can move the focus into an icon and press the MENU (right) pointer button to get the *Window Menu*, which operates as described below.

6.2.2 Transferring the Focus with Keystrokes

In AT&T's implementation of the OPEN LOOK Window Manager, you can move from one window to the next by the key sequence ALT+F6. You can move to the previous window with Shift+ALT+F6. In the OpenWindows version of the OPEN LOOK Window Manager, use Next Application (Alt-n) or Previous Application (Alt-Shift-n). This only works if you have the olwm.KeyboardCommands resource set to Full; see Appendix K, *OPEN LOOK Mouseless Operations*

6.2.3 What to do if olwm Dies and the Focus is Lost

It may occasionally happen that the window manager will "drop out from under you." While it is an important and reasonably well-debugged program, it is possible for *olwm*, like any computer program, to fail under certain rare circumstances. In this case, you will see a brief flurry of action as all the titlebars disappear and any iconified windows pop open.

Fortunately, in most cases the input focus reverts to X11's native "point to type" mode, so you can just move the cursor into any terminal emulator window and type the command *olwm&* to get the OPEN LOOK Window Manager back. For those cases when the input focus is left in click-to-type mode and *not* in a terminal window, you can use the *Secure Keyboard* feature of *xterm* to get the focus back, as described in Appendix A, *The xterm/olterm Terminal Emulator*. Of course you have to have an *xterm* running to use this mode. Alternately, you can log in at another terminal, or remotely, and restart *olwm* manually. Or, if you have a File Manager running, pull down *File -> Custom Commands -> UNIX Shell...* which will try to start a shell window for you; from there, start *olwm* as before†.

6.3 The Workspace Menu (or Root Menu)

The overall screen area of your display is called the *Workspace* in OPEN LOOK terminology, or the *root window* in X11 terminology. Most window systems give you a special "master menu" when on this area, and OPEN LOOK is no exception. Moving the pointer into the background window and pressing the MENU pointer button causes the *Workspace Menu* (also called the *Root Menu*) to appear. The normal content of this menu is shown in Figure 6-1.

Figure 6-1. OpenWindows Workspace Menu

Both *Programs* and *Utilities* have a menu mark beside them, so selecting either one will bring up a menu. *Properties* has a window mark ("...") after it, to indicate that selecting it will bring up a pop-up window, the properties manager. The Properties Manager (or Workspace Manager) is described in Section 14.1, "Properties Resource Editor" on Page 331. Both *Help...* and *Desktop Intro...* give access to tutorial documents using the *helpviewer* application described in Chapter 7, *The OpenWindows DeskSet Clients*. The former is an

† The final act of desperation before rebooting a machine is, if you have a shell window but can't get the input focus into it using any of these techniques, use copy-and-paste to copy "o", "l', "w", "m", and end-of-line into a shell window!

overall table-of-contents handbook, as shown in Figure 2-6, and the latter is the Introductory tutorial that appears in an uncustomized OpenWindows session. And selecting *Exit...* will bring up a confirming dialog; if you click on *Exit* in the dialog, you will terminate your X11/OPEN LOOK session.

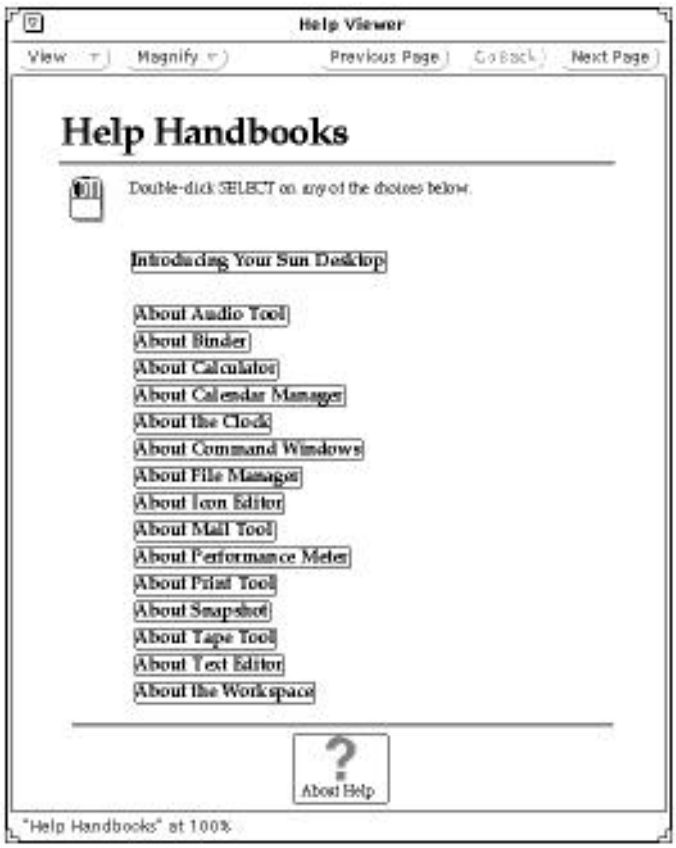


Figure 6-2. Top Level Help Handbook

Figure 6-3 is the *Programs* menu:

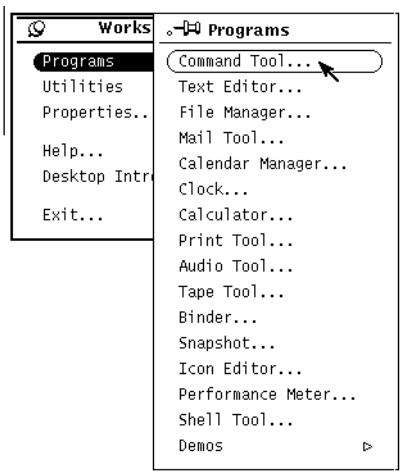


Figure 6-3. Workspace Menu, Programs submenu, OpenWindows Version

Selecting any one of these will start up the appropriate application. For example, selecting *Command Tool* will start up a *cmdtool* terminal emulator, described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*. The individual programs are discussed throughout the remaining chapters of this book. Most of them are described in Chapter 7, *The OpenWindows DeskSet Clients*. The *Utilities* menu is presented in Figure 6-4.

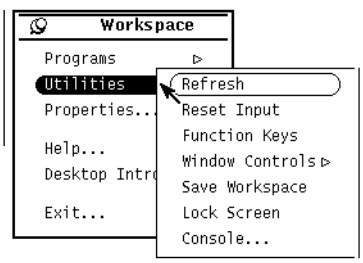


Figure 6-4. Workspace Menu, Utilities submenu, OpenWindows Version

The *Refresh* item will cause each window, including the root or Workspace window, to be re-painted or re-displayed. This is normally used when system output has inadvertently appeared over top of windows, for example, when console output is directed to the screen instead of to a terminal emulator such as *xterm -C* or *contool*...

The *Reset Input* item (which only appears on old versions of OpenWindows under SPARC, not on Intel x86 systems) has to do with the NeWS system.

The *Function Keys* brings up the Function Key map, as shown in Figure 6-5. See the *vkbd* man page for details..

Window Controls is described below.

Figure 6-5. OpenWindows Function Key Popup

The Sun version of *Save Workspace* will save the names and locations of your applications in a file *($HOME/.openwin-init*) so that anytime you later start OpenWindows you will have your own customized set of clients. However, it may miss certain MIT X applications or those developed with non-OPEN LOOK toolkits. It may be preferable to customize your.*xinitrc* file instead, which is portable to more versions of X. See Section 3.4, “Customizing the X Environment: Specifying Resources” on Page 69.

Lock Screen starts a screenlock program; it is used when you will be away from your workstation or X terminal for a short period; you don’t want to log out of X, but you don’t want other people to use your account while you’re away. While the screen is locked, nobody can type into your windows. To resume normal activities, you must hit RETURN and then correctly type your login password.

The last item, *Console*, starts up a terminal emulator *(Cmdtool*) with console output directed to it. This is useful to prevent further need for the *Refresh* item described above.

✗ A word of warning: the *Console* entry (as well as *contool* or *xterm -C*) should only be used on (run from) a workstation, not on a file server. Older versions of UNIX would allow any user to take over the system console output function, and your console window would then get all the operator-specific messages such as full filesystems, login problems, etc.

The Window Controls menu is shown in Figure 6-6.

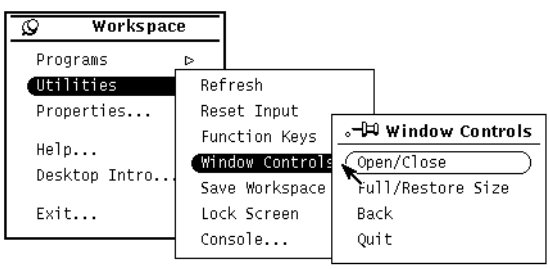


Figure 6-6. Workspace Menu, Window Controls Submenu

The *Window Controls* allow you to operate on one window or on a group of windows that you have selected. You can select a group of windows in two ways:

- Sweep out a rectangle on the Workspace that totally encloses the windows or icons you want

- Click SELECT on the titlebar, frame, or icon of the first, and ADJUST on the titlebars, frames or icons of the rest

Once you've done this, you can *Open/Close* them as a group, which will open any that are closed, and close any that are open. You can use Full/Restore Size, which will make any that are normal size occupy the full height of the display, and any that are full height return to their normal height and position. You can move the whole group to the *Back*. And if you want to clean up your screen quickly, you can *Quit* a whole group of windows. Note that the group association is temporary; it is broken as soon as you use SELECT for any other purpose on the same display.

This is the default set of menu items. On Sun's OpenWindows, you can change the Workspace menu or even completely replace it; on AT&T's OPEN LOOK you can add items to the *Programs* submenu. This is described in detail in Chapter 13, *Customizing olwm*.

6.4 The Window Menu: Moving, reshaping, and iconifying Windows

As with most window systems, you can use the titlebar to perform common window operations, such as moving, resizing, and closing the window. Unlike some other X11 window managers, however, OPEN LOOK lets you use the entire frame around the window to do these operations, so you don't have to have the titlebar visible. In the two-dimensional monochrome OPEN LOOK, the frame is very thin on the sides and bottom, but you can find it with a bit of practice. If you watch the screen cursor carefully, it "blinks" briefly as you move the pointer onto the frame around a window. You can set it to a larger thickness using X Resources, shown in the section *"Customizing the X Environment: Specifying Resources."*

In color OPEN LOOK, the frame is normally painted in a color that contrasts with the background and with other windows, so it's easier to see.

What can you do in the titlebar or frame? Use the MENU button to get the *Window menu*. The normal OPEN LOOK Window Manager *Window menu* has the items shown below:

Table 6-1

Choice	Function	Shortcut
Close	Selecting this choice will cause the window to be 'closed' or 'iconified', that is, replaced by a tiny icon window that represents the window.	Click SELECT on triangle at left of titlebar.
	(Clicking MENU on the icon will get a Window menu with the word 'Close' replaced by 'Open', which lets you open the window).	Double-click SELECT on icon

Table 6-1

Choice	Function	Shortcut
Full Size Normal Size	This will make the window assume its maximum size, which usually means the full height of the screen. If the window has already been set to "Full Size", this selection will read "Normal size," and selecting it will return the window to its original size.	Double-click SELECT on titlebar
Move	Sets up for mouseless (or mouseful) *move* operation.	Click and drag SELECT on titlebar
Resize	Sets up for mouseless (or mouseful) *resize* operation.	Click and drag SELECT on any resize corner.
Properties	This is not used at present. It will someday allow you to access the application's Property sheet, if it has one, from the Window menu.	L3 key
Back	Move this window to the back (bottom) of the stack of windows.	L5 key
Refresh	Re-display the window, in case some bits of it got clobbered by accident.	
Quit	Causes the window and the program or application controlling it to be terminated.	Shell window: ^D or `exit`.

If you've previously used Motif or Microsoft Windows, note that "Close" here means *iconify*, and does not terminate the program, while "Quit" means terminate. Motif and MS-Windows use the term "minimize" for *iconify*, and use "Close" to mean terminate.

There are several common operations that do **not** appear in the *Window menu*. For example, there is no *Front* item, because you can move any window to the top (front) of the window stack by clicking SELECT in its titlebar or frame. Since an icon is a (small) window, you can move an iconified window to the top just by clicking SELECT in it once (remember that clicking SELECT twice will cause the window to be opened). As well, the Sun keyboards have a *Front* (L5) key which has the same effect.

While there is a mouseless *Move* item on the *Window* menu, most users prefer to move windows around by pressing *and holding* the SELECT button on the titlebar or frame. When you do this, an outline of the window will follow the pointer to show the new position of the window. When you release the SELECT button, the window is actually moved. Similarly, you move an icon by clicking and holding SELECT on the icon. To use the mouseless *move*, click SELECT on the *Move* item, then use the arrow keys on the keyboard to move the window around. To move more quickly, hold down the SHIFT key while pressing the arrow key.

Similarly, there is a mouseless *Resize* item on the *Window menu*, but most users seem to prefer using the pointer. To change the size or shape of a window, move the pointer so that it is over one of the four "resize" corners. Notice that the cursor changes from an arrow to a circle, meaning that you can move in any direction at all. Again, a "rubber band" or outline will show the new size as you move, and when you release, the window will actually change size. To use the mouseless mode, click SELECT on the *Resize* item, then use the arrow keys to adjust the size of the window. Holding the SHIFT key will cause faster resizing. Most programs nowadays will react correctly to being resized. What about those that don't? For some versions of the *vi* editor, you should not resize the window while *vi* is running; if you do this, you must quit *vi* and restart it. For older systems, there is a program called *resize* that sets up the information on the resized window. It is described in the reference page in Part Three of this guide.

As mentioned under the description of "Window Controls" above, you can perform many of these operations on groups of windows. Say you wanted to open several iconified windows at the same moment. Click SELECT on the first one, then click on the other(s). Note that each one has its border highlighted to show that it is part of the selection. When you've SELECTed the last one, press MENU, and release on OPEN. With a mighty flash, all the windows you've selected will open at once. Another method of grouping is to click and hold SELECT on the background (root window), and drag SELECT until the rubber band box completely encloses all the windows you want to group. Now moving any one of them will cause them all to move. In either case, the association between these windows lasts only until you use the SELECT button to select any other items on the screen.

OpenWindows offers the following shortcuts, using extra keys that appear on the Sun Type-3 and Type-4 keyboards:

Alt-T Pressing the T key while holding the Alt key toggles the input focus between the last two windows that were selected.

L5 Pressing the L5 key when the pointer is in a window moves that window to the top (front). If the window is already at the top, L5 moves it to the bottom (back).

L7 Pressing the L7 key when the pointer is in a window iconifies the window; pressing it when the pointer is in an icon restores (opens) an icon to its window.

6.5 The Virtual Desktop (Virtual Edges)

The OPEN LOOK specification allows a window manager to provide "virtual edges" to the workspace (or root window). This allows the workspace to appear to be larger than the physical screen. However, neither supported version of *olwm* provides this at present. It is left to a contributed program called *olvwm*, derived from the Sun *olwm* source code, to provide a facility that, while not conforming to the OPEN LOOK specification for "Virtual Edges", provides a more useful solution to the problem of limited screen "real estate". *olvwm* is especially useful on small, low-resolution screens (800x600, 1024x768, ...) in which you can't usefully show much more than one full-sized terminal emulator window at a time. But it is also worthwhile on normal- and high-resolution screens (1152x900 and up). In fact, if your office desk looks anything like mine on a typical day, *olvwm* can make

your workstation screen *really* look like your desk, bringing the paperless office one step closer to reality.

The following description of using *olvwm*'s virtual desktop facility is adapted from the manual page by Scott Oaks, who added the virtual desktop facility to *olwm* to produce *olvwm*. The full reference page is reprinted in Section Three of this guide. You can obtain *olvwm* by anonymous *ftp*, as described in the *Preface*.

Olvwm starts by displaying a virtual desktop manager (VDM) window along with any other tools normally started in your *.xinitrc* file. The virtual desktop manager is a reduced display of all windows active on the desktop at any time. The area of the desktop which is visible (that is, which appears on your screen) is outlined in the virtual desktop manager. For example, Figure 6-7 shows a normal VDM that has room for six screen-sized segments. The one in the upper left has a large terminal emulator labelled "darian". The one below it has a single window called "Snapshot" (see Chapter 9, *Graphics Clients*). The top-middle screen has two *Mailtool* windows (see Chapter 7, *The OpenWindows DeskSet Clients*). The window under that has one large *Xfig* window (a contributed graphics program).

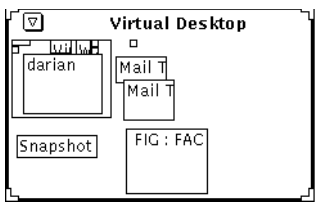


Figure 6-7. olvwm's virtual desktop manager

But what are those tiny boxes at the top? They are icons, clocks, and other gadgets. Figure 6-8, continuing the example, shows the screen in the upper left corner:

6.5.1 Moving windows with OLVWM

You move windows on the screen just as you normally would in *olwm***.** You can also drag windows from the screen into the virtual desktop manager or from the virtual desktop manager to the screen, or just within the virtual desktop manager. Note that if you select on a window's frame which is overlapping the virtual desktop manager, you will end up dragging the window into the virtual desktop manager, and it will suddenly shrink to the scale of windows in the VDM. You can disable this with an X resource (`AllowMoveIntoDesktop`, see Chapter 12, *Setting Resources*) if you are bothered by this behavior.

If you select a window inside the virtual desktop manager (but not necessarily in the current segment), you can move the window. This is just like moving a window around on the workspace, except that it all happens in much-reduced scale.

If, instead, you double click on any area inside the virtual desktop, then that segment becomes the current segment.

Figure 6-8. The top left screen of OLVWM's virtual desktop

6.5.2 OLVWM Sticky Windows

The Virtual Desktop Manager window never moves on your screen when you change views. That's because the desktop manager window is permanently "sticky." "Sticky" windows never move position on the screen when you change your view into the desktop. To set a particular window as sticky, simply select *Stick* in its frame menu. You may similarly *Unstick* a sticky window via its menu.

6.5.3 Advantages of OLVWM

The most obvious advantage is that you have more space to work on; windows that you aren't using can be moved off the screen without iconifying them.

Another advantage is that it allows the Workspace Menu to be automatically pinned up; this feature is not in the standard *olwm* yet.

6.5.4 Limitations of OLVWM

There are two problems with *olvwm* that affect NeWS programs (such as FrameMaker 3.1 OPEN LOOK) and SunView programs running under OpenWindows.:

- NeWS programs built with The NeWS Toolkit use the NeWS Window Management functions, which look almost exactly like *olwm*; under *olvwm*. SunView programs use the SunView mouse conventions, described in Appendix L, *Running SunView Applications on OpenWindows*. These programs always appear on the current segment of the screen, and cannot be moved to other segments; their *Window Menu* does not have the *Stick* or *Unstick* options. That is, they are in effect sticky; they always appear even if you try to move away from them.

- Some NeWS programs *sometimes* behave as though click-to-type focus policy were in effect, even though pointer-focus policy is in effect.

Users of such programs should consider using *olwm* instead.

6.5.5 The Future

It remains to be seen whether *olvwm* will be "adopted" by Sun, or will remain in the contributed software category. At any rate, the facility if offers is sufficiently useful that Sun ought to provide it somehow.

6.6 The Workspace Manager (Properties Manager)

The OPEN LOOK specification calls for a Workspace Properties window to allow the user to change "global preferences". This is implemented by a separate program from the Window Manager, called the Properties Manager. It is mentioned here because it often seems to be part of the OPEN LOOK Window Manager. Indeed, it cooperates with *olwm* in that some of the property changes it makes are acted upon immediately by the Window Manager. However, it is a separate client program started by the Window Manager. Accordingly, the Workspace Properties Window is described more fully in Chapter 14, *Customization Clients*.

Note that this is not the same as the Properties item on the *Window* menu of individual windows; that item is disabled if it is not yet implemented, as in OpenWindows.

We have now discussed the most important features of the OPEN LOOK specification and the OPEN LOOK Window Manager. We now turn our attention to a detailed discussion of the important clients, starting with the terminal emulators. The *cmdtool/shelltool* terminal emulator clients are discussed in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, while the pre-OPEN LOOK terminal program *xterm* is discussed in Appendix A, *The xterm/olterm Terminal Emulator*. Subsequent chapters discuss other important clients.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 7

The OpenWindows DeskSet Clients

One advantage of the OPEN LOOK GUI is that it specifies several basic clients such as the File Manager described in Chapter 4, *Using the OPEN LOOK File Manager*. In addition, Sun's OpenWindows provides quite a collection of useful clients above and beyond what the OPEN LOOK specification requires. The name for this collection is "The DeskSet". Table 7-1 lists the DeskSet clients plus some additional clients that are part of OpenWindows even though not in the DeskSet.

Table 7-1. DeskSet and other Sun Clients

Program	Part of?	Function	Described in:
answerbook	unbundled	Hypertext document reader	page 152.
audiotool	DeskSet	Sound play/edit/record	page 152.
binder	DeskSet	Describe file types, icons, etc.	page 155.
calculator	DeskSet	Desk Calculator	page 155.
Catalyst CDware	Catalyst distribution	Lots of demonstration programs, Catalyst catalog, etc.	page 156
clock	DeskSet	Clock accessory	page 157.
cm	DeskSet	Fancy calendar program	page 158.
cmdtool	OpenWindows	Terminal emulator	Chapter 5, *The cmdtool/shelltool Terminal Emulator*
dbxtool	SPARCworks (was DeskSet)	Interface to *dbx*	page 166; also SPARCWorks documentation

Table 7-1. DeskSet and other Sun Clients

Program	Part of?	Function	Described in:
filemgr	DeskSet	File Manager ("dekstop manager").	Chapter 4, *Using the OPEN LOOK File Manager*
helpviewer	DeskSet	Documentation viewer	page 166.
iconedit	DeskSet	Icon editor	Chapter 9, *Graphics Clients*.
mailtool	DeskSet	Multimedia mail system	page 166.
pageview	OpenWindows	PostScript previewer	Chapter 9, *Graphics Clients*.
perfmeter	DeskSet	Performance load monitor	page 175.
printtool	DeskSet	Print system interface	page 177.
searchit	unbundled	Full text searching	
showme	unbundled	Interactive conference	page 180.
snapshot	DeskSet	Screen shot grabber, raster file viewer	Chapter 9, *Graphics Clients*.
tapetool	DeskSet	Interface to *tar*	page 180.
textedit	DeskSet	Text editor	Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

7.1 answerbook

Sun's *Answerbook* is a complete collection of the operating system documentation, plus several books including a few from this series, on a single CD-ROM with searching software. You can locate material by keywords, or by book name. You can set your own "bookmarks" to find material that you have located. And you can print any portion of the documentation on a PostScript printer.

As the *Answerbook* is an unbundled product, we don't provide a full description of it in this book. However, we commend Sun for making the documentation available on CD-ROM, and recommend that at the very least each workgroup acquire a copy of the *Answerbook* corresponding to the version of SunOS that they are using.

7.2 audiotool

The Sun SPARCstation has a microphone connector and an audio-quality loudspeaker. One can infer from this that its designers foresaw the need for multi-media computing in the late 1980's. *audiotool* is a program that lets you record your voice into a sound file, play sound files, and edit sound files. It is used automatically if you try to send or receive

a sound file while using *mailtool*, described in Section 7.13, “mailtool – the OpenWindows Mail Interface” on Page 166.

When you first start *audiotool*, you get a window like that in Figure 7-1, and (assuming you have your microphone connected properly as per your model of SPARCstation’s documentation), you are immediately ready to begin recording and playing sounds. Without a microphone, you can of course play existing sound files, but not record new ones.

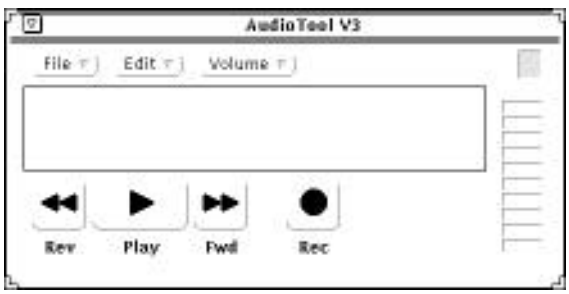


Figure 7-1. AudioTool initial screen

You can load an existing sound file from the `File` menu’s `Load` item. This load item has a directory browser in it. A useful directory is */usr/demo/SOUND/sounds*, if you installed your SunOS with the demonstration files. Just type this directory’s name in the Name field and press RETURN, and you’ll be able to view the directory, as shown in Figure 7-2

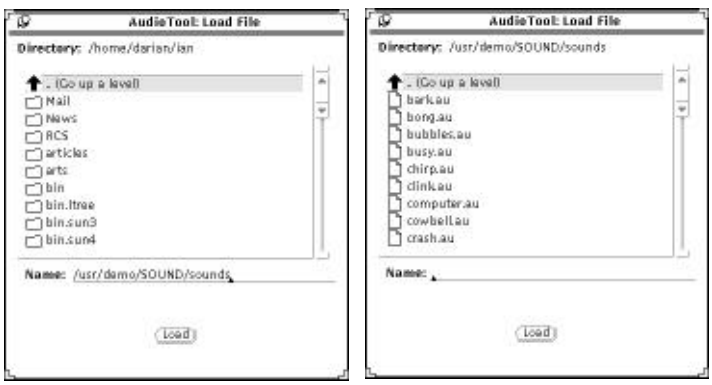


Figure 7-2. AudioTool File Browser: Initial state, and in demo directory

We can’t really describe the sounds here, but they are fun to listen to. When you select a file and click the `Load` button, or just double-click SELECT on a filename, it will be loaded any played.

The drag-and-drop target in the upper right corner allows you to drop in a sound file from the File Manager or any other source; it will be loaded and played.

You can also control playing from the main window shown in Figure 7-1, where the four large glyph buttons (labelled "Rev", "Play", "Fwd" and "Rec") are intended to model the control buttons on a standard audio tape recorder. Clicking SELECT

- on the `Play` button begins playing, and advances the position indicator as it plays. The `Play` button is replaced by a `Stop` button.
- clicking on the Stop button stops playing or recording, and the `Play` button reappears.
- clicking on `Rew` (rewind) takes you to the beginning of the current sound image,
- clicking on `Fwd` (fast forward) moves you to the end.
- clicking `Rec` begins recording, and the `Rec` button becomes a `Stop` button while recording.

While it is playing, the sound level meter at the right side of the window displays the relative sound level, like a "Vu meter" on an audio tape deck. The length of the current sound image is shown in the right footer of the main window, and convenient time markers are shown below the sound graph in the main window. You can also control the output volume and the output destination (internal speaker or the external jack) from the `Volume` menu's popup windows, as shown in Figure 7-3.

Figure 7-3. AudioTool Volume Pop-up

The `Edit` menu on the main window lets you access, from a menu, such functions as `Cut` and `Paste`, used to rearrange bits and snippets of sound. You can play at this forever if you're not careful! This menu also gets you at the `Properties` sheet, which controls such global functions as whether sounds should be played automatically when loaded, and whether "silence detection" (removal of dead space from sounds) should be performed. You can also change the directory into which temporary files are saved, in case your */tmp* filesystem fills up while you are editing a complicated sound sequence.

If you have sound files in other formats, you may wish to consider the free-software program *sox* a library of files for converting sound files from one format to another. Convert the files to Sun *.au* format, then load them with AudioTool. To send sound files to another user, see *mailtool*, described on page 166.

7.3 binder

The *binder* program is used to control the Classing Engine routines that classify files to decide what icon to display for them, how to print them, and other information. The Classing Engine is used by most of the DeskSet tools, including the OPEN LOOK Window Manager. However its greatest user is the File Manager, so it is documented in Section 4.4.4.1, “Binder: Customizing the File Manager in OpenWindows” on Page 100.

7.4 calctool – the OpenWindows Calculator

Calctool is a scientific calculator. Unlike *xcalc* described in Chapter 8, *calctool* does not emulate a specific brand or model of calculator, but tries to provide the best features of several types in one convenient program. It has fixed, scientific and engineering display modes, basic, financial, and scientific modes of operation, ten register memories, ten user-definable functions, and zillions of other features. Here is a look at its keyboard:

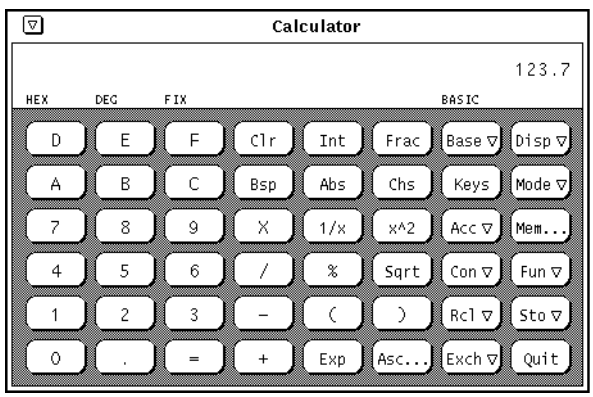


Figure 7-4. Calctool in action.

Most keys have keyboard equivalents. Let’s face it, moving the pointer around to press keys on a window-based calculator may be fun for a minute or two, but just about anybody can type faster than they can point-and-click, particularly for entering numbers. For example, to divide a 95-acre farm among three children, just type

```
95 / 3 <RETURN>
```

with the input focus on *Calctool*, and you will see in the display window that each will inherit 31.67 acres.

When you are typing, you can use normal OPEN LOOK cut and paste operations to put data into the window or copy it out. However, the text field does not show that it has been selected; this is probably a bug that will be fixed by the time you read this. For example, to copy the calculation from the text of the previous paragraph into *calctool*, I just triple-clicked on the “95 / 3” line, hit the COPY key (L6), moved the pointer focus to the Calculator window, and hit PASTE (L8). I then pressed <RETURN> and got the answer. OPEN

LOOK was used to copy the answer back into that paragraph, too: I selected the "31.67" in the *calctool* window by clicking SELECT and dragging the pointer across all four digits (ignoring the fact that text didn't get highlighted as I dragged), hit COPY, moved back into the *cmdtool* window in which we were editing this chapter, and hit PASTE.

In keeping with OPEN LOOK's context-sensitive help facility, each key has a short help menu, as shown in Figure 7-5, that shows the name, keyboard equivalent, and function of the key.

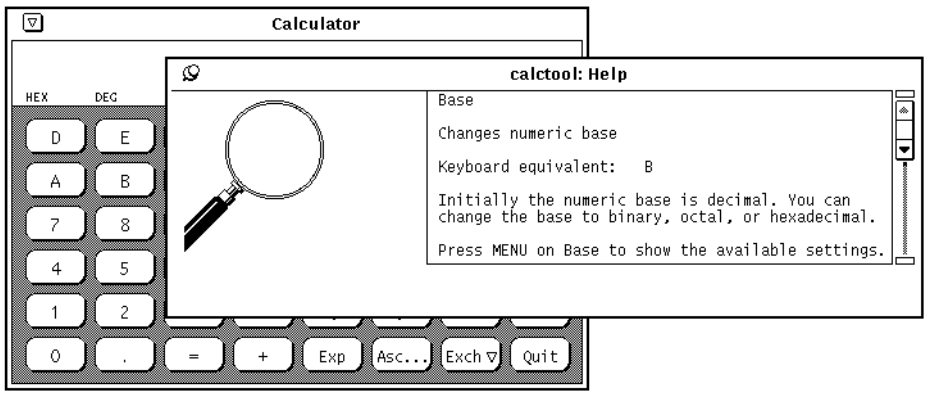


Figure 7-5. Calctool help on "Base".

Each button with a menu mark (such as *Base*, *Acc* and *Con* in Figure 7-4) features a menu. You can pull down the menu by pressing and holding the MENU pointer button, or pop it up by pressing and quickly releasing MENU. Or you can apply the default action from the menu just by clicking SELECT on the menu button. There are literally dozens of functions; too many to detail here. They are all explained in the reference manual page in Section Three, and summarized in the on-line help for each key.

It also has Financial and Scientific modes, and lets you define your own functions.

7.5 Catalyst CDware

One of the best ways to find out about third-party software is to "try before you buy." Sun-Soft produces a CD-ROM filled with demonstration copies-many of`them unlockable-of a large number of commercial software products`covering publishing software, database, imaging, games, software`development, and many other areas. All products are distributed

in a standardized OPEN LOOK browser format as shown in Figure 7-6, which shows a few of the products listed in Volume Four of CDware.

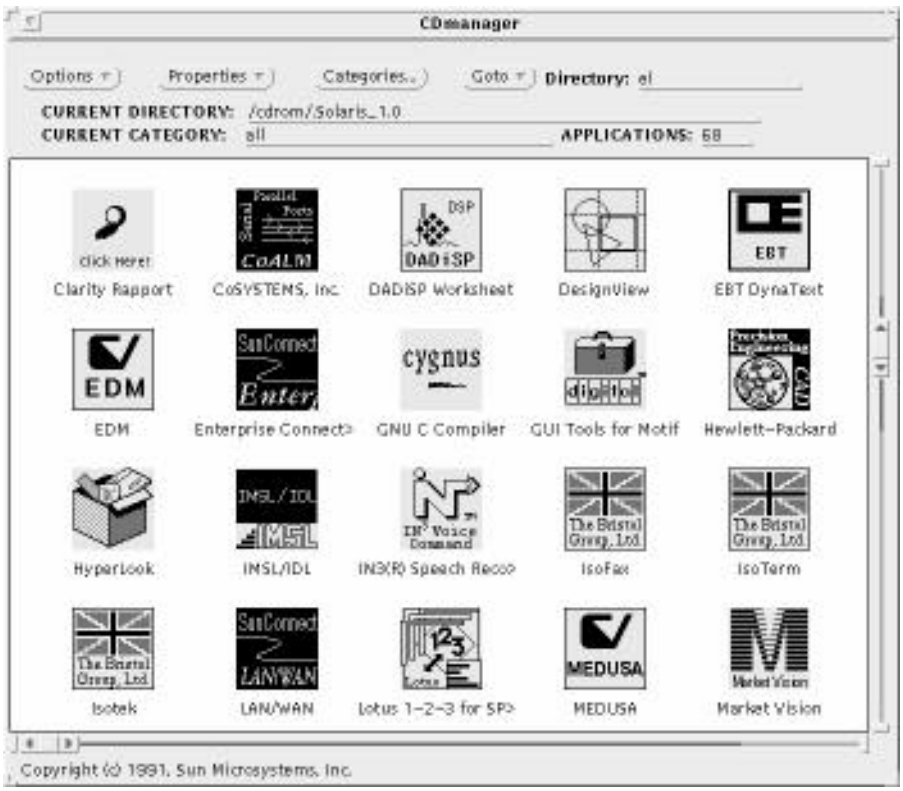


Figure 7-6. Cdware browser and installation program

The CDware CD is issued once or twice a year and is distributed free of charge to Sun owners and users. To get a copy, contact your Sun representative, or SunSoft Inc., Attn: Catalyst CDware, 2550 Garcia Avenue, Mountain View, CA 94043, U.S.A..

7.6 clock

The *clock* program is a simple rectangular clock. It has a Properties dialog that lets you select analog or digital display, analog or roman display in the Icon window, 12 or 24 hour mode, and include the seconds and the date if you wish. The MIT clock programs are described in Section 8.1.1, “Clock Programs: xclock and oclock.” The ultimate discussion of X clocks is the book *X User Tools*, where an entire chapter is given over to the many varieties of clock that you can run under X.

7.7 cm – the OpenWindows Calendar Manager

One of the most common examples of a useful window system program is that of a calendar tool. Most calendar utilities (even the two-decades-old UNIX *calendar* program) let you specify your appointments, and be notified about them in advance. Some of the screen-based ones let you display a week, month, or year, and print calendars in various formats. But most of them ignore the fact that people who work together often need to meet together. Very few such tools let you schedule meetings interactively, by looking at various users' calendar files. Sun's *cm*, or Calendar Manager, is one tool that does so. When initially started, it comes up iconified with the current day and month shown, as in Figure 7-7:

Figure 7-7. Calendar Manager initial icon.

To open the icon, just double click SELECT, or press L7, with the pointer on the icon. It opens up to a large month-at-a-glance format:

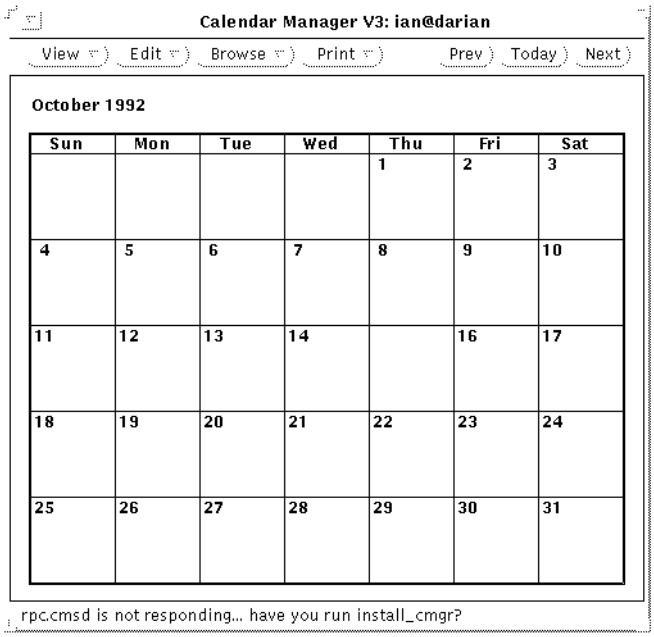


Figure 7-8. Calendar Manager main window.

As you can see, this has a menu bar at the top and displays the current month in the main panel. We'll discuss the menus presently, but first let's look at the most common operations.

Let's say we want to schedule an appointment for lunch on the 27th of this month. We need only move the pointer into the calendar's box for the 27th, and double-click SELECT, to start up the Appointment Editor window:

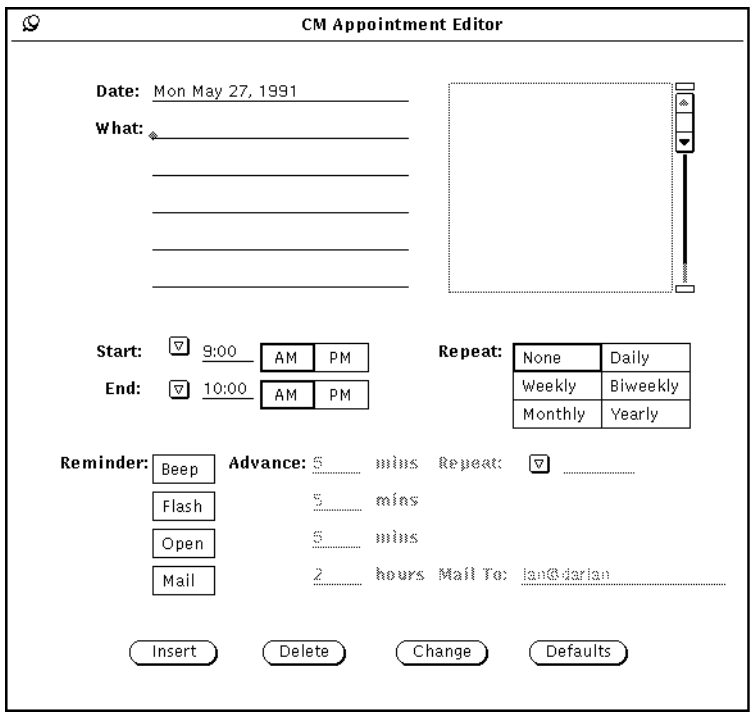


Figure 7-9. Calendar Manager Appointment Editor.

We can type a note about the appointment into the *What* field, and the times in the Start and End fields. As you'd expect from an OPEN LOOK application, there is a context-sensitive Help facility for **all** controls in the window, accessed by pushing the HELP (F1) key. For example, if you need information about the *Start* and *End* time fields, you might press HELP on the Start field, and you will see a display similar to that in Figure 7-10:

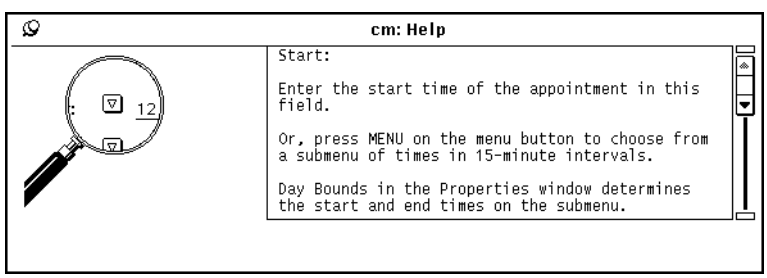


Figure 7-10. Help on the Appointment Editor.

Once you've entered the note about the appointment, the times, you can activate any of the four warning types:

Table 7-2. Reminder Methods for Calendar Manager Appointments

Method	Meaning
Beep	Causes the X display's bell to ring the specified number of minutes in advance of your appointment. Adjust the time to give yourself enough time to get there!
Flash	Causes the *cm* window or icon to start flashing the specified number of minutes before your appointment.
Open	Causes the *cm* icon to open into a window the specified number of minutes in advance.
Mail	Causes *cm* to send electronic mail the given number of hours in advance. You can direct the mail to another user or account if someone else normally manages your time, or if you are using someone else's account.

Once you've set up the appointment the way you want, click SELECT on the *Insert* button. Notice that the appointment now appears both in the scrolling list on the Appointment Editor window *and* in the full-month view:

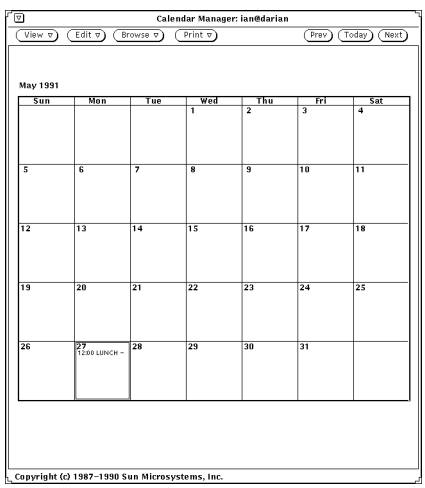


Figure 7-11. Appointment now appears in Month View.

You can also use the scrolling list to select an appointment for alteration or deletion (click SELECT on its entry in the list and then on the *Delete* or *Change* button.). You can recall the default values (usually to start installing a new appointment, since the default *What* item is "nothing") just by clicking on the DEFAULTS button in the editor.

Back on the main window, there are two sets of buttons in the menu bar. The left side has the traditional *View*, *Edit* and other buttons; the right side has *Prev*, *Today*, and *Next buttons*, The *View* menu lets you control the view in the main panel; you can see just one day, or the week, or a whole month (the default), or even a whole year. *Edit* will by default get you into the Appointments Editor described above; it can also be used to start the Properties sheet that lets you change:.

Table 7-3. Calendar Manager Property Choices

Choice	Meaning
Editor Defaults	Set the defaults for the various Reminder types described above.
Day Boundaries and View	Controls the time that the day starts and ends (adjusted with two OPEN LOOK sliders), and the default view (Year, Month, Week, or Day).
Browser Calendars	Lets you specify a list of other people whose calendars you want to inspect.
Access List and Permissions	Lets you give other people permission to browse, insert, and delete items in your calendar.

Most of these are straightforward. The last, Access List and Permissions, deserves an example. By default, everyone ("the world") can inspect your calendar file:

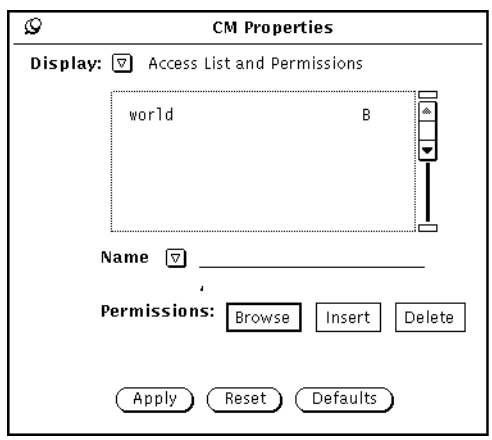


Figure 7-12. Calendar Manager Access Control Screen.

Suppose we want to give user “mary” on machine “gamut” the ability to schedule appointments for us. We add her name (using normal “user@host” notation) and turn on Insert and Delete, then click on the menu mark beside the word *Name* to Add it. This abbreviated menu pulls down to a list of Add, Delete or Change; the default is Add, so you can just click on the menu mark, or hit RETURN after her name, as a keyboard shortcut for Add.

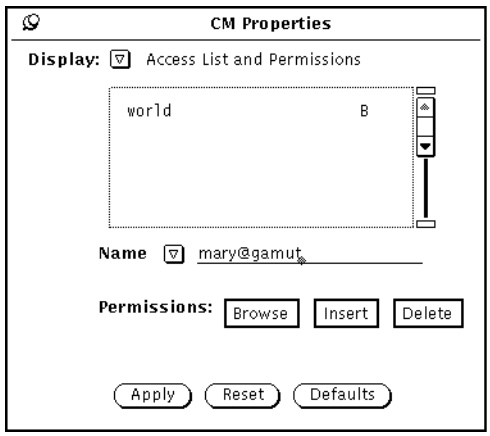


Figure 7-13. Giving Mary Insert and Delete Access.

After adding all the entries we wish to add, we click APPLY to make them permanent.

The *Browse* item lets you browse in others’ calendars, assuming they have not turned off the default world Browse permission. It gives you either a scrolling list of people you commonly browse, or lets you select them from the menu. Browsing is similar to working with your own calendar, except that you can see several calendars at once if you invoke the multi-browse feature

The *Print* menu item lets you print any of the standard views (day, week, month, or year), or set the printer properties. For example, Figure 7-14 is a printout of the month of January,

1999. Normally it is printed in landscape mode to fill a full "US Letter" size page; we've rotated and reduced it to fit this book:.

2001

04/03/95 02:50 PM ian@sqrex

January

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

February

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28			

March

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

April

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

May

S	M	T	W	T	F	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

June

S	M	T	W	T	F	S
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30

July

S	M	T	W	T	F	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31				

August

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

September

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

October

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

November

S	M	T	W	T	F	S
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

December

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Year view by Calendar Manager

Figure 7-14. Calendar Manager Printout of Month View

The *Prev*, *Today* and *Next* buttons move you forward, backwards, or to today's entry; however, the entry will be displayed in whatever view you have selected. For example, if we have set the View to be Year, and click on Today, we get the current year. If we click on Next enough times, we can get the view shown in Figure 7-15: †

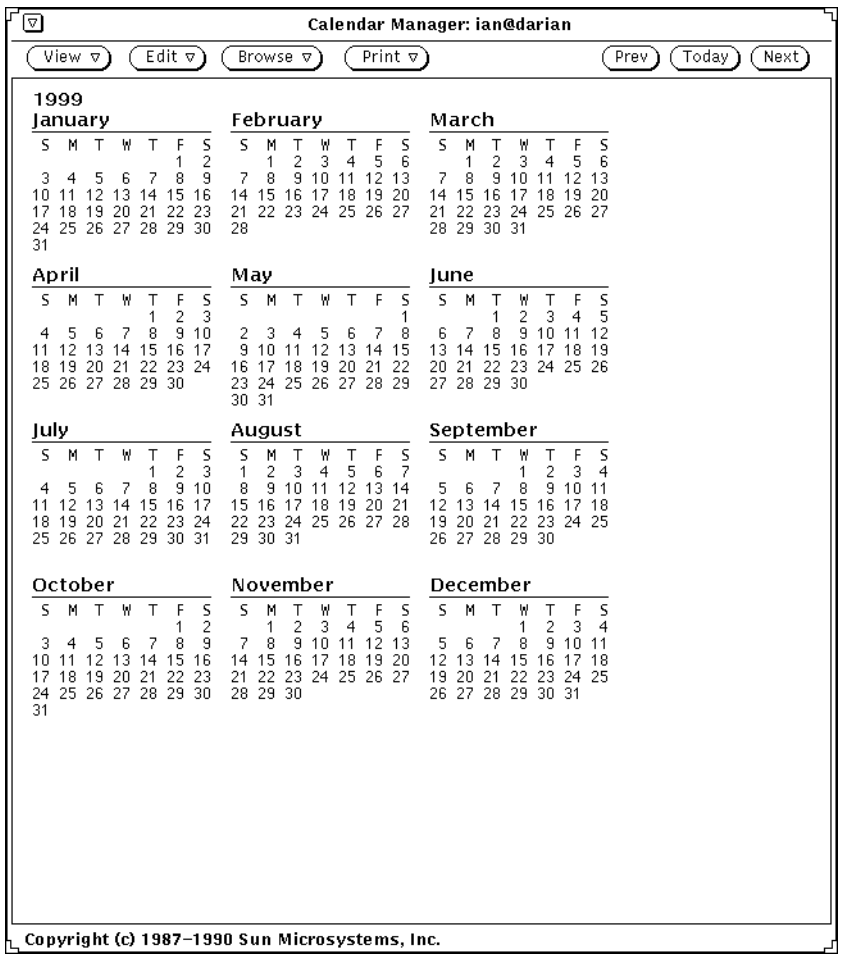


Figure 7-15. Full-Year Calendar.

7.7.1 Related Programs

The contributed software program *xcalentool* provides similar functions, It does not have any capability to inspect other users's calendar files. It does have a simpler file format, and has an "include" file capability so that public lists of holidays or other key dates can be

† The program will stop at 1999; it will not display the last year of this century or any years in the next. Presumably later versions will allow you to plan further into the future.

included without making multiple copies of the dates files. It is not afraid to print year calendars into the twenty-first century. Another contributed program is *xcal*.

7.8 cmdtool

The OpenWindows standard terminal emulator is *cmdtool*, which has been described at length in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

7.9 dbxtool, debugger

The *dbxtool* debugging interface originated as a SunView program in early releases of SunOS. The first X-based version was included with Release 2 of OpenWindows. With Release 3 it was moved from the DeskSet into the unbundled compilers package known as SparcWorks (ProWorks on the Intel version of Solaris). With Release 2.1 of Solaris, this moving target has been renamed *debugger*. Since it is in any event a tool for programmers, it is not documented in this Guide; consult the appropriate Sun documentation.

7.10 filemgr

The File Manager is a visual interface to your files and programs; this type of program is also called a "desktop manager". We have devoted all of Chapter 4, *Using the OPEN LOOK File Manager*, to this program.

7.11 helpviewer

The *helpviewer* program displays a "help handbook" and allows you to view it, move around, print pages, etc. It is started automatically when a new user starts up OpenWindows the first time (see Figure 2-6 on Page 33), and every time thereafter until the user removes it from their start-up files as described in Section 3.5, "Customizing your Session Start-up" on Page 72.

```
More detail here.
```

7.12 iconedit

The *iconedit* program is for creating and modifying the icons that represent files and running programs. It is described in Chapter 9, *Graphics Clients*.

7.13 mailtool – the OpenWindows Mail Interface

Mailtool is an OPEN LOOK replacement for the standard Berkeley *Mail* program. Berkeley *Mail* (known as *mailx* in System V), when started up in mail-reading mode, provides you with a menu of your mail items:

```
Mail version SMI 4.0 Tue Nov 7 10:46:02 PST 1989 Type ? for help.
           "/var/spool/mail/ian": 3 messages 3 new
>N 1 rab@research.att.com Sat May 25 14:20 20/632 S-Plus Review
 N 2 To tim@ora.com Sat May 25 14:23 43/1904 comments on chapters
 N 3 root Sat May 25 14:24 16/458 Output from "at" job
Mail>
```

For each mail message you see its status (N for New), message number, sender's name and date/time, size in lines and characters, and the subject. Single-letter commands (s for save, r for reply, etc.) let you work individual messages in your mailbox.

Mailtool tries to give you all the flexibility of the underlying Berkeley Mail program within the framework of the OPEN LOOK user interface. When started normally, it shows up as one of two icons, depending on whether you have any unread mail or not:

Figure 7-16. Mailtool icon when you have no mail.

Figure 7-17. Mailtool icon when new mail has arrived

When you open the icon, any mail you do have is displayed in a menu format similar to that shown in Figure 7-18:

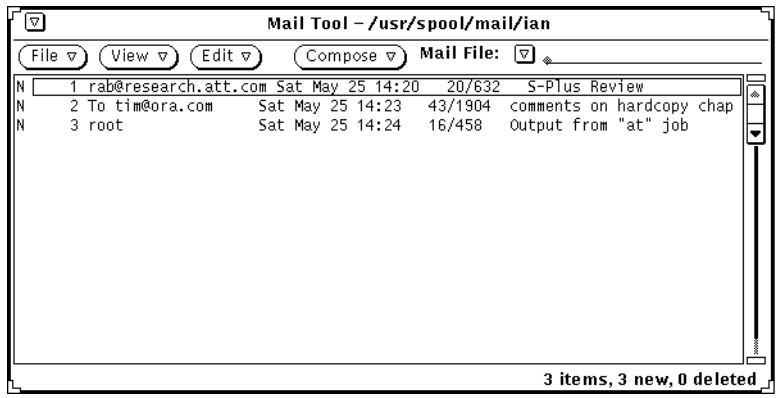


Figure 7-18. Mailtool main window with three messages to be read.

To read a mail message, just click SELECT on it (the "menu" is really an OPEN LOOK Scrolling List). Double-clicking on a message, or clicking once on the message then on the *View* button, causes the message view window to appear, with the selected message in an (editable) text window:

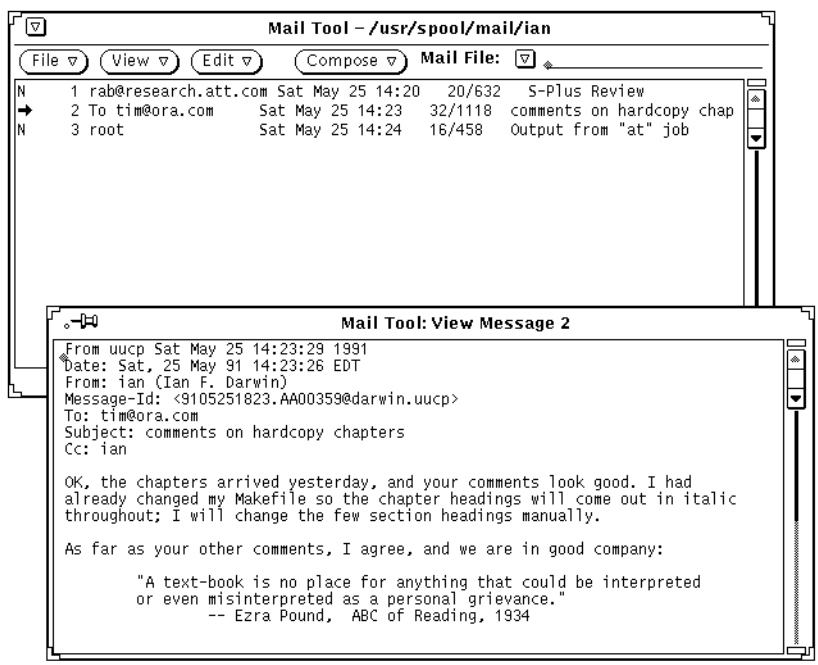


Figure 7-19. View a message.

This window can be scrolled in the usual way, and the text can be edited as described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

7.13.1 Sending a message

Should you decide that a reply to this message is in order, or if you wish to compose a new message, pull down the *Compose* menu. The main items here are *New*, which starts a new message, and *Reply* which replies to the sender of the message that you are viewing or have selected. If no message is selected, you can only compose a New message. The *Compose* window looks like Figure 7-20:

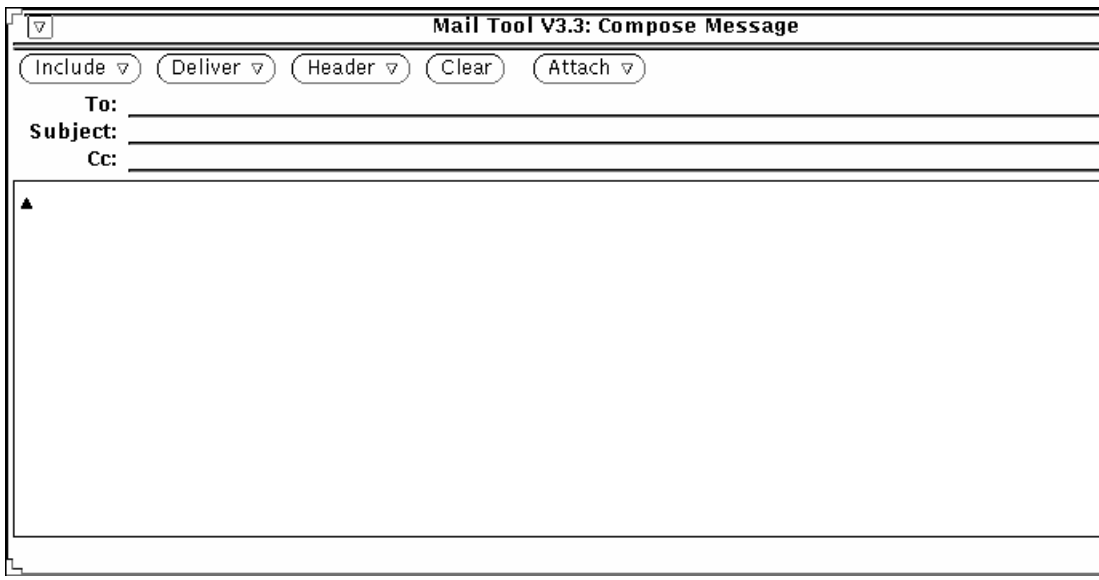


Figure 7-20. Compose Window

There are some controls at the top; for now, just type the name of the recipient into the text field labeled *To:*. The main part of this window is a standard OPEN LOOK text screen, as described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, into which you can type the body of your message. When you're happy with the message, just click SELECT on the *Deliver* button at the top, and it will be sent.

7.13.2 More Menus

The *File* menu lets you *Open* new mail, *Move* or *Copy* selected messages into a holding file, *Print* the selected message(s), *Save changes* that you've made, and be *Done* with (e.g., iconify) the *Mailtool* program.

The *View -> Messages* menu item lets you see either the full header or an abbreviated header, as controlled by the *Ignore* line in the *.mailrc* file in your home directory. This is useful to suppress the display of uninteresting "message headers" inserted by most mail transport agents such as *sendmail*. For example, if we have the line

```
ignore Received Message-ID Status
```

in my *.mailrc* file, then the abbreviated message view (which is the default) might look like this:

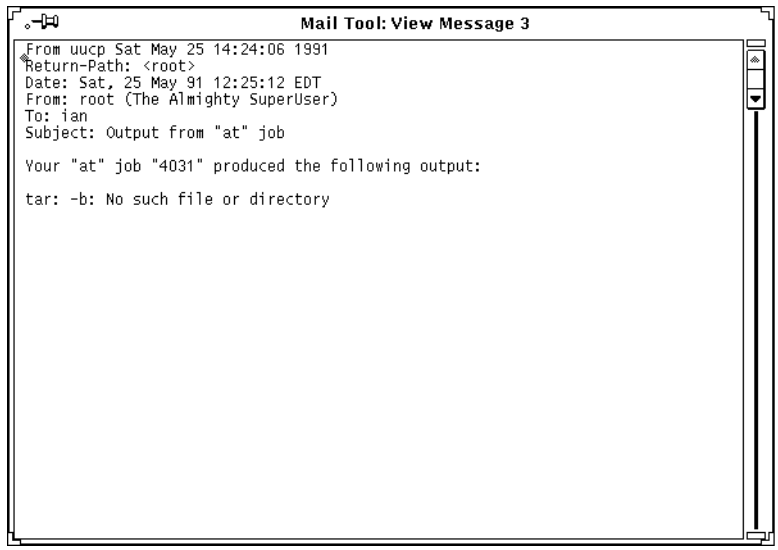


Figure 7-21. Abbreviated Message View.

The Full message display of the same message would look like this:

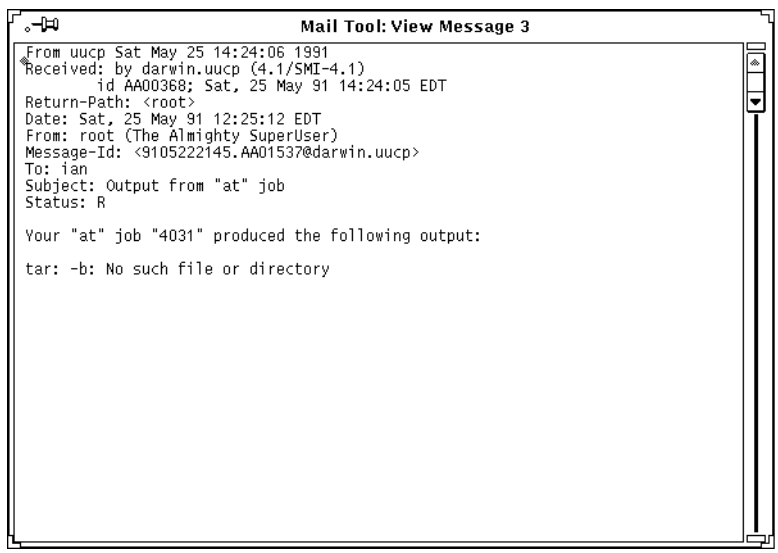


Figure 7-22. Full Message View.

The *View* menu also lets you select the *Next* or *Previous* message. And it also has a *Sort* item for sorting the messages; this item seems to be permanently disabled in this release.

The *Edit* menu you lets you cut, copy, delete or undelete messages. It also gets you access to *Find*, which tries to locate a person by using Sun's NIS (formerly YP) password service (you can't use this facility if the machine you're running *mailtool* on isn't served by NIS). And it gets you the *mailtool* Property sheet

The *Compose* menu on the main window allows you to create a *New* or *Reply* message. As previously noted, this pops up the *Compose* window, used for entering mail messages. You can also *Forward* a message, which lets you use an existing message as the basis of a new message. And finally, this menu gets you into a mode where you can control the *vacation* mail program. This will send people mail stating that you are on vacation (or whatever message you like) and that you will read your mail when you get back.

7.13.3 Attachments

The *Mailtool* program allows you to include or "attach" files, audio clips, and other items. When you receive a message with attachment(s), you will see them listed in a file-manager-like window at the bottom of the *View Message* window. When composing a message, you can select the menu item *Include->Show Attachments* to create a similar window at the bottom of the *Compose* window. This window has its own *File* menu that lets you Add a file, and an *Edit* menu that lets you rename or remove attachments. But since this is a file-manager type window, you can more easily add objects just by drag-and-drop from any filemanager window. *This is the best way to send a copy of a file to the recipient of your message*

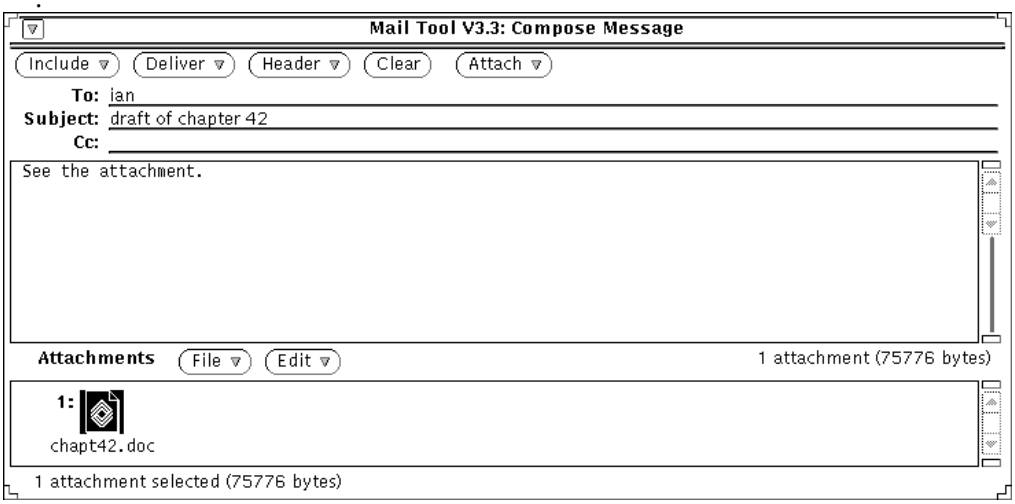


Figure 7-23. Compose Window with one Attachment

The *Compose* window also has an *Attach* menu, which lets you attach a voice message using *AudioTool* or an Appointment from *Calendar Manager*.

7.13.4 Menus in Compose Window

The *Compose* window is used to compose new mail and reply to incoming messages, and has several menus at the top. *Include* lets you include the currently selected message (it does not default to the message you are replying to; you must select a message). By default, this button includes the message *bracketed* by the conventional Internet-style messages:

```
----- Begin Included Message -----
----- End Included Message -----
```

Alternately, you can have the message *Indented* by one tab position.

The next menu button is to *Deliver* the message. This normally terminates the *Compose* window; you can also iconify it, leave it up, or leave it up and clear the message, by choosing options from this menu.

Using the third button, you can add *Header* fields such as CC (carbon copy), BCC (blind carbon copy, i.e., the recipients don't see the BCC recipient's name), and other fields (which you specify usingthe Properties window as described in Section 7.13.6, "Customizing Mailtool" on Page 173).

The next button, *Name Finder*, starts up the same message finder described above under *Edit -> Find*. It relies on NIS to scan your NIS domain's password file looking for a name that matches the name you specified. This feature is not included in later versions of the software.

The final button, *Clear*, is the measure of last despair: it clears the entire compose message window and lets you start all over again. If you have text in the window, of course, you will be prompted if it is alright to discard the text.

The *Compose* window is a baseframe, so you can keep working on it while you've iconified the rest of *Mailtool*.

7.13.5 Saving and Printing Messages

Mailtool provides a facility for saving all your mail messages in a subdirectory. You can specify this in the Mail Filing category of the Properties window (See "Customizing Mailtool" below).

The main menu scrolling list allows you to use the OpenWindows shortcut for CUT, L10, to delete the current message, even while there is a View window active for it. Just move the pointer into the main window and hit the CUT key. The message will be deleted, the next message will be selected, and the new message will appear in the View window if there is one active.

Since the main menu is a scrolling list, you can select any item in it for drag-and-drop operations. Just click SELECT to select the item, release the pointer button, then click SELECT again without moving the pointer, and hold SELECT down while dragging. If you drag the message (an envelope icon follows the pointer) onto a *printtool* icon (described in *Section 7.16, "printtool – the OpenWindows Printer Interface"* later in this chapter), the

file will be sent to the printer. (In older versions you then had to select *Edit -> Undelete* to keep the message in your mailbox, since the drag and drop operation removed it from your mailbox). You can also use use *File -> Print* from the main window to print the message that is currently displayed.

As another possibility, you can save the message in a file just by dragging it into a File Manager (see Chapter 4, *Using the OPEN LOOK File Manager*) window and dropping it into a directory. Since mail messages don't have names, the saved message will have a name made up for it by *mailtool*. Depending on what version of the software you have, the saved message will initially have a filename of either *Mail* plus a large pseudorandom number, or of *sun-deskset-message*. You can rename it using the file manager (just click on the name, rather than the icon, type your new name, and press Enter). Another way to copy a message into a file is to use *File -> Edit -> Save New File* in the window menu of the *mailtool* View window.

7.13.6 Customizing Mailtool

There are two main ways of customizing mailtool, the *~/.mailrc* file and the *Edit->Properties* menu item, which pops up a Properties window.. For the most part you should avoid editing your ~/.mailrc file and use the Properties window. The categories of customization are shown in Table 7-4:

Table 7-4. Mailtool Properties Customization

Category	Meanings
Header Window	Message Listing Window: Retrieval frequency, notification, geometry, delivery, custom buttons.
Message Window	Lines to display, print command, hidden fields
Compose Window	When composing text: marker for included text, file to log outgong messages to, defaults, custom fields for *Header* menu
Message Filing	Mail folders to appear in *Load*, *Copy* and *Move* menus
Template	Neat feature - lets you compose or reply with form letters!
Alias	Updates user and group aliases (also used by *)*
Expert	Miscellaneous Options

One customization that can only be performed by editing the *~/.mailrc* file is the display format used in the Header Window. There can be any of a number of *variables* in this file. One variable is *showto*, which controls whether address of the "To" person or recipient will be displayed. Another variable is *header_format*, which specifies the format of the header. This variable is rather like the C language printf format string, in that a percent sign fol-

lowed by (usually) one character specifies the formatting of one item. The formats which can be used are shown in Table 7-5:

Table 7-5. Mailtool Format Codes

Format Code	Result
%c	number of bytes in the message
%C	number of bytes in the message contents (excluding header)
%f	entire From field
%l	number of lines in the message
%L	number of lines in the message contents
%i	Message-ID field
%m	Message Number
%s	Subject: field of message
%t	To: field (recipient)
%r	Recipient (sender, or to line if you are the sender)
%d	date and time of the message
%n	sender's real name (if any, otherwise same as %r)
%?header?	Contents of *header* field, if present
%%	Print a real % character

For example, the default format is something like

```
header_format="%3m %-18.40f %16.16d %4l/%-5c %-.100s"
```

where a number means the width of a field, a period means "no more than", and a negative sign means to left-justify the field.

7.13.7 Other Mail Programs

The standard MIT X11 client *xbiff* will show you whether you have mail or not, using an icon similar to that an old version of *mailtool*. And the contributed program *faces* will give you a fancier display of incoming mail, showing a pictorial representation of each sender.

The three mail messages shown in the examples above might appear like this with *Faces*: 9z

Figure 7-24. Faces display of incoming mail.

But neither of these has the ability to display your mail; they just tell you that you have mail to read.

The standard MIT X11 client *xmh* provides a sophisticated mail reading interface to the MH mail system; both MH and *xmh* are described in detail in the O'Reilly Nutshell book *MH & xmh: E-mail for Users & Programmers* by Jerry Peek.

7.14 Pageview

The *pageview* PostScript previewer is described in Chapter 9, *Graphics Clients*.

7.15 Perfmeter

The *perfmeter* program is useful for showing the "load average" and other measures of busy-ness on one or several machines. It is simpler to use and more flexible than *xload* (see Chapter 8, *Other Standard Clients*). The *perfmeter* client knows how to reach out across the network to ask another machine for its load average (or any of almost a dozen statistics). Its default is to show the percentage of CPU time that is used, rather than the load average; you can get the load average just by adding `-v load` on the command line. Let us say that you want to watch the load average on both the local machine (call it *kansas)* and a server named *oz*. Assuming that both machines are running the standard RPC daemon *rstatd(8C)*, you need only type this set of commands in a *kansas* xterm:

```
perfmeter -v load & perfmeter -v load oz &
```

The result might look like this:

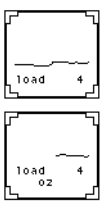


Figure 7-25. OpenWindows Perfmeter running locally and on Oz

As you can see, *perfmeter* has the advantage (over the older *xload* program) that it gives a numerical scale to its graph, rather than the numberless graph that *xload* offers. As well, *perfmeter* lets you select CPU load, disk I/O, paging, ethernet traffic, and many other

things to watch just by pressing the *Props* key (L3) or by pulling down a menu in its window. Its *Properties* panel lets you select two or more load factors to monitor, and lets you change several other attributes of its operation. Figure 7-26 shows the Properties window, which you get by selecting Properties from the application menu in the *perfmeter* window.

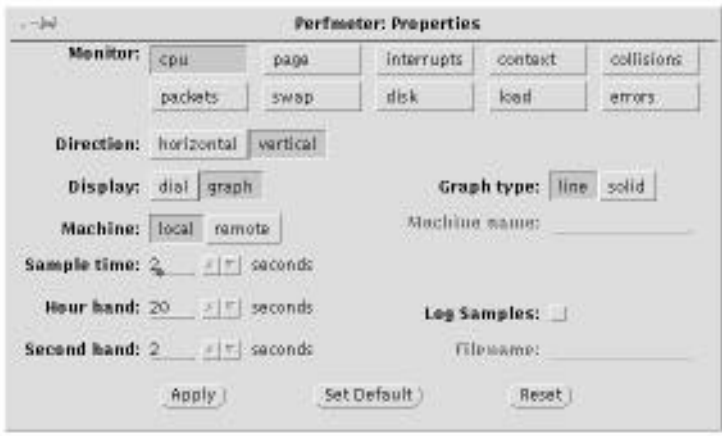


Figure 7-26. Perfmeter Properties Window

You can select any of the load indicators that you want:

See the reference manual page in Section Three of this guide for more information about either *xload* or *perfmeter*.

7.16 printtool – the OpenWindows Printer Interface

Printtool provides a simplified OPEN LOOK-style interface to the line printer system. If you start it, either by typing the command *printtool* or by selecting it from the *Programs* menu, you get a start-up screen similar to this one:

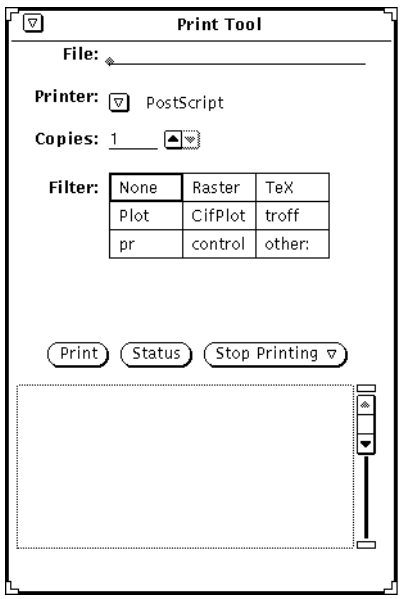


Figure 7-27. Printtool main window.

The main function of this tool is to send files to the printer for you. There are two simple ways to do this. One is to have a copy of printtool sitting iconified, drag a file's image from the *filemgr* program, and drop it onto the icon for *printtool*. The other way is to have a copy of *printtool* open, type the filename in the *File* text field, and click on the *Print* button.

Other options let you choose the printer to use, specify the number of copies, and specify what type of data the file contains. The defaults are: the first printer found in the file */etc/printcap*, one copy, and no special "filtering." As well, push buttons let you *Print* a file (like the *lpr* command), get the *Status* of all jobs (like the *lpq* program), and cancel or *Stop Printing* either one job or all the jobs you have queued (like *lprm*).

To specify an alternate printer, you can use the OPEN LOOK abbreviated menu beside the name of the default printer. Just click MENU on the menu mark beside the word **Printer:**, and you will see a list of the available printers. Select one, and it will become the current printer. The first one in the list is the default, so you can always return to the first one just by clicking SELECT directly on the menu mark.

To change the number of copies, you can either click on the up arrow to add one to the current number, or the down arrow to subtract one from the number. For more drastic changes, say 54 copies, you naturally don't have to click the up-arrow 54 times. The number display

is just an OPEN LOOK text field. Move the pointer to the right of the current number, click SELECT, type Control-U to clear the field (or DEL to delete one character), and retype your number.

The Filters listed are normally used only with special types of data files. One exception is *pr*, which paginates your text file before printing it, using the standard UNIX utility of the same name.

The *Print* button causes the named file to be sent to the printer with the currently selected options. The *Status* button causes a listing of the current print jobs to appear in the small scrolling window at the bottom of the page. If you want to delete one or more print jobs, just select them from the scrolling list, and select *selected Print Job* from the *Stop Printing* menu button. You can also cancel all your outstanding print jobs on this printer just by selecting *All Jobs* from the same button.

In short, printtool gives you control over your printers. For more details, consult the reference manual page in Part Three of this guide, as well as the *lpr* manual page in your normal UNIX documentation.

7.17 SearchIt - full text searching

Searchit is an unbundled product from SunSoft that lets you build an index of a collection of files, then rapidly search the index to locate files. We used it in building the OPEN LOOK

version of this book. Figure 7-28 is an example of searching for all documents containing the phrase “escape sequence:”

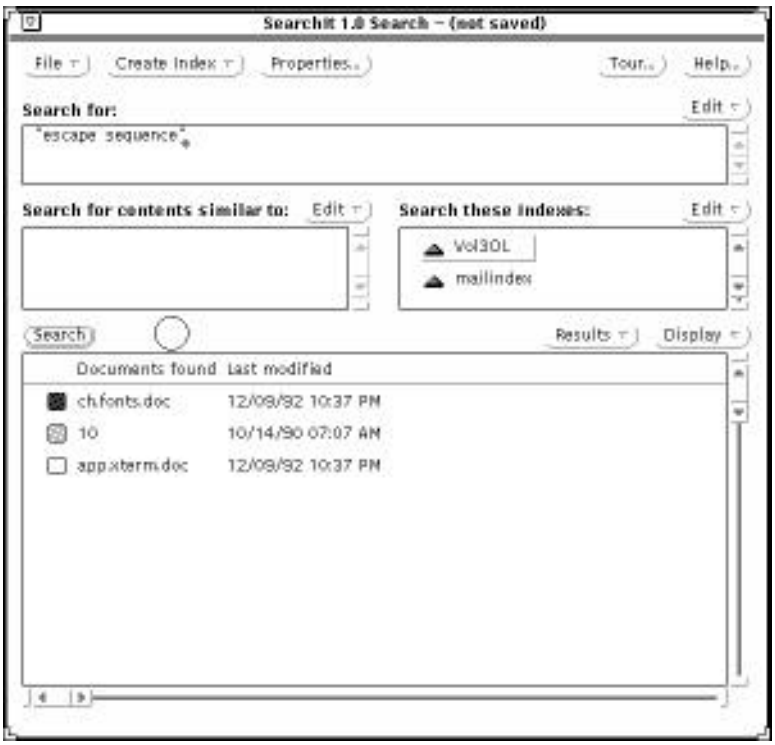


Figure 7-28. Searchit in action

This search found three documents; the first, shown with a black icon, was judged most likely to meet our search criteria, the second, shaded gray, less likely, and the third, shown with a white icon, least likely. Double=clicking on any of the files will bring up a *viewer* window, with command and menu buttons to jump to the next occurrence of a search word, the next document, etc., All in all it’s a good product.

7.18 ShowMe - graphical conferencing

ShowMe is an unbundled converencing tool from SunSoft. It is very easy to use, and will communicate with another or several other copies of itself, one per workstation display. Figure 7-29 is a (hypothetical) example of using *ShowMe*, .

Figure 7-29. Part of a ShowMe conference

The figure definitely looks better in color, so do look at the Color Plates section. Each participant in the conference is given a different color, and anything they add—text, freehand drawing, and outline or filled rectangles or ovals—appears in their color. We like this program, as it is a good example of making computers useful with an intuitive interface.

Figure 7-29 is adapted from our review of this program in the January 1993 issue of *SunExpert* magazine. If you need a "shared whiteboard" facility, we recommend you investigate this program. You can get an evaluation copy without obligation by contacting your Sun representative.

7.19 tapetool – the OpenWindows Tape Interface

Tapetool provides a simplified interface to the standard tape archive utility *tar*.†

.Because *tapetool* is an interface to an existing program, not all the messages are very detailed. Pay close attention to the "footer" section of the main window, and also look in your Console window for messages. For example, trying to write on a tape that is not write-enabled will product the rather uninformative message

```
Write terminated
```

in the window footer; the more detailed messages

```
st0: tape is write protected /bin/tar: can't open /dev/rst8 : Permission
          denied
```

will normally appear in your console window (or in the shell window from which you started *tapetool*). Furthermore, this program will probably never have 100% of the flexibility that the UNIX command line interface to *tar* has. For example, there is no way (except for reverting to a terminal emulator window and issuing an *mt* command) to make *tapetool* read the second or third tar file on a single tape. Nor is there any way to support extra options that local or contributed software versions of *tar* may offer. In fact, you can't use *tapetool* with the Free Software Foundation's contributed version of *tar*; it just doesn't work, due to some incompatibility. However, for simple uses, *tapetool* provides access to the 90% of *tar*'s functionality that is most commonly needed.

When you invoke *tapetool*, you get a simple main window that looks like Figure 7-30:

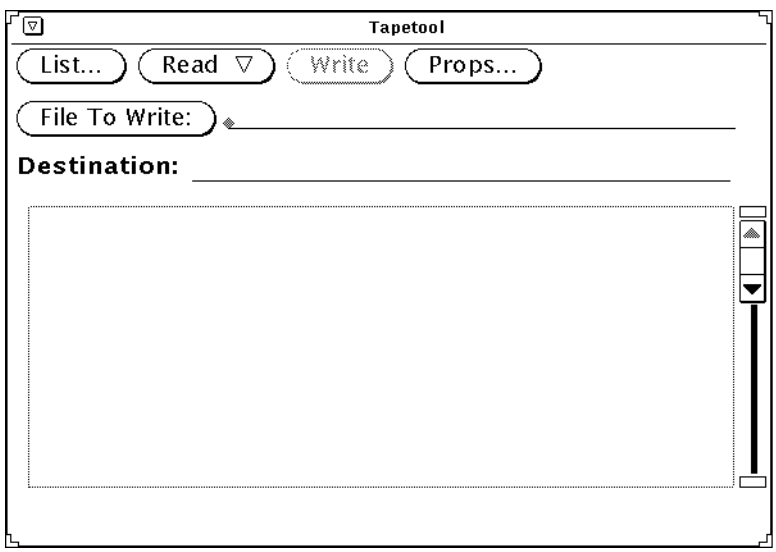


Figure 7-30. Tapetool main window.

As you can see, the main choices are to *List* the tape, *Read* files from it, or *Write* files to tape, and there is also a property sheet. Let's look at each of these operations.

† It does not work with, and in its present form cannot be made to work with, other incompatible tape programs such as *cpio*, *dump*, etc.

7.19.1 Listing and Extracting tape files

Assuming a tape is mounted in your system's default tape drive, just clicking on *List* will generate a list of files on the tape, in a separate window, like so:

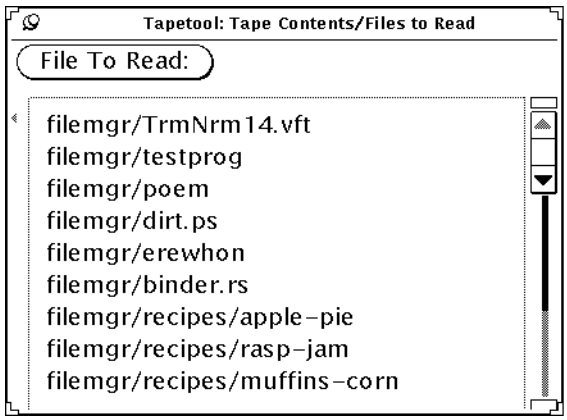


Figure 7-31. Tapetool list of files.

Quite often, obtaining a listing of the files on a tape is a prelude to extracting some or all of them. But given that the list of files is in the computer already, in the "list of files" window, it would be an awful waste of personpower to re-type some of the filenames. And to make maximum use of having the list in the computer, you should be able to specify the files you want in two ways: either by selecting some from the list to be read or extracted, or by selecting some to be excluded (an "all but these files" extraction). *Tapetool* does provide both methods of specifying the files. That display of the list of files is an OPEN LOOK *Scrolling List*, so you can select individual files from it. To extract individual files, just select them — they'll be highlighted by a box around each item — then pull down *Read* from the main window and release or select on *Selected*. In fact, if you have a list displayed, that is the default, so you can just click SELECT on the *Read* menu button. You also have to select a directory *into* which the files are to be extracted, in the *Destination* text field. If you want the files read into the directory in which you started *tapetool*, just type "dot" ('.'), meaning the current directory. Otherwise, type the path to the directory where you want the files places. Then select *Read/ Selected*.

The *Read* item *Entire List* implies that there must be a way to get a partial listing of the tape. Indeed there is. Get a listing of the tape, as above. Select the files that you do **not** want to extract (remember that you can click and hold SELECT and drag it across a scrolling list to quickly select a bunch of files for exclusion), then press the MENU button in the Listing window, and select *Delete Selected* to discard the ones you don't want, leaving those you do. As a shortcut, you can hit the CUT key, L10. Then you can select *Read -> Entire List* and have the program read onto disk just the files you want.

7.19.2 Writing Files with Tapetool

To write a file or directory onto a tape, you must mount the tape with write enabled. For the common QIC cartridge tape drives used on Sun systems, this is accomplished by turning the small plastic arrow in the upper left of the cartridge so that the arrow points to the left, i.e., away from the word "SAFE." For half-inch (reel-to-reel) tapes used on older Sun servers, putting the small plastic write-enable ring, flat side out, into the back of the tape reel enables writing.

Then you have to select the files you want to write. For each file, just type its name in the *Files To Write* textfield, and press RETURN. *Tapetool* will verify that the file is readable and, if so, add it to the scrolling list in the main window. Once you're happy with the list, you can just click SELECT on the *Write* button in the main window. Be *very* careful to watch for error messages. In fact, you should probably make it a habit to always get a listing of a tape (as described above) after writing it, just to be sure that all the files really were written.

7.19.3 Setting Tapetool Properties

The property sheet is used to specify the *device special file* of the tape drive in case the default drive isn't suitable, and to specify some common *tar* options. It lis shown in Figure 7-32:

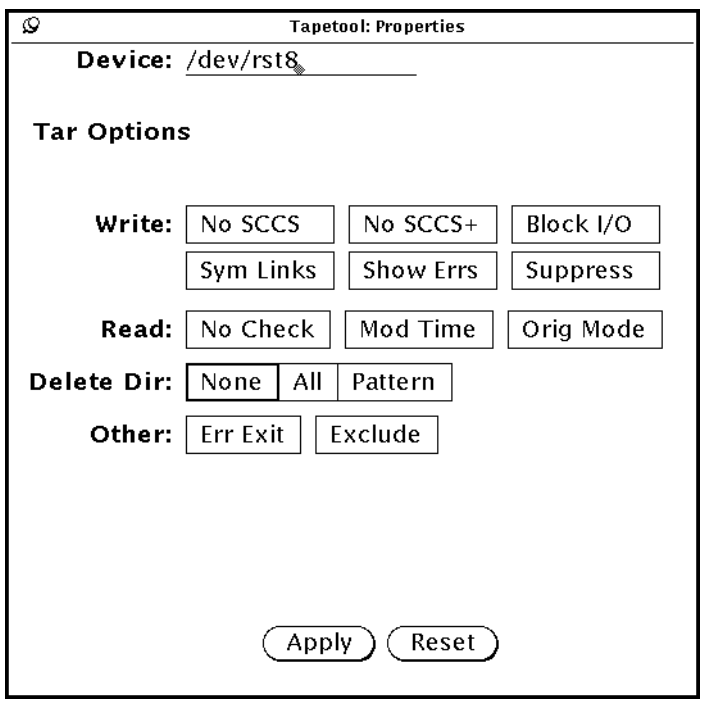


Figure 7-32. Tapetool properties.

If you have more than one tape drive, or use multiple densities, you probably already know the names of the device special files to use. If not, consult your system administrator or Sun's manual pages on the particular tape driver you are using. Note that although some versions of the *tar* program allow you to use a remote tape drive, this option does not seem to be supported at present (maybe in OWN Release 3)?

The other options control specific options of the *tar* program; please refer to the reference manual page for this program in your version of the SunOS reference manual to see exactly what they do. If you are already familiar with *tar* you probably recognize many of these, and if not, you can safely skip them for the time being. There is also a reference page for *tapetool* in Part Three of this guide, which summarizes these options in a briefer format.

7.20 textedit

The *textedit* editor has been described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

7.21 Demonstration Programs and Games

Sun has always been an innovative company, and employs a lot of talented people that delight in innovation. The NeWS system, though it has been phased out from current releases of SunOS,, demonstrated conclusively that PostScript is a useful language for interactive display management. Some of these demos and games are in NeWS, and will not work after OpenWindows Version 3.2..

You can access most of these from the *Demos* item on the main workspace menu, assuming you have the full OpenWindows package installed. That menu is shown in Figure 7-33:

Figure 7-33. OpenWindows Demonstration Programs Menu

Table 7-6 is a list of these programs and a few notes on them.

Table 7-6. OpenWindows Demonstration Programs

Name	Function	Notes
Icosahedron	Bouncing geometric shape	From MIT dist'n.
Maze	Generates and solves random mazes	Sun.
Muncher	Geometric progression	From MIT dist'n.
Plaid	Random plaid patterns.	From MIT dist'n.
Puzzle	Solves child's 4x4 puzzle	
Worms	Colorful worms escaping.	
Xsol	Solitaire game.	
Spider	Double-deck Solitaire game	
Jet terminal	Terminal emulator.	See page 135.
Jed	Text editor.	See page 135.

Table 7-6. OpenWindows Demonstration Programs

Name	Function	Notes
NeWS Clock	Another clock.	Sun, NeWS.
FontView	Detailed look at characters in a font.	Sun
Pixelview	PostScript drawing demo	Sun, NeWS
Circles	Circles demo.	
Colors	Choose colors with sliders.	Buggy!
RasterRap	Amazing operations on raster images.	Neat! Try doing this in raw X11...
OLIT Sampler	Sampler of the OPEN LOOK GUI components (controls, etc.)	See "Other Controls" on Page 41
TNT Sampler	As above, but for (obsoleted) TNT toolkit.	NeWS.
Text	Text demonstration	See Section 10.5.3, "The "text" demo program (OpenWindows up to 3.2)" on Page 279.

7.22 Summary

We have now covered the standard DeskSet client programs that are part of Sun's OpenWindows package.We now look at the standard MIT clients that are part of all X11 distributions, in Chapter 8, *Other Standard Clients*.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 8

Other Standard Clients

This chapter gives an overview of other clients available with X, including window and display information clients, printing utilities, the *xkill* program, and several "desk accessories." It also examines some of the features common to applications written with the X Toolkit.

The standard X11 distribution includes many useful clients. Most of these are included in OpenWindows Release Three. We have grouped these X11 clients according to basic functionality:

- Desk accessories: *xclock*, *oclock*, *xcalc*, *xbiff*, *xload*, and *xman*.
- Text editor: *xedit*.
- Window and display information programs: *xwininfo*, *xlswins*, *xlsclients*, *xdpyinfo*, and *psps*.
- Programs to remove a client window: *xkill*, pam.

In many cases we refer you to OPEN LOOK or OpenWindows alternatives described elsewhere in the book. Most sections in this chapter are intended to acquaint you with the major features of some of the available clients. Additional detailed information is provided on the reference pages for each client in Part Three of this guide.

Many of the standard clients ere written using a programming library called the X Toolkit. As explained in Chapter 2, the X Toolkit includes a set of predefined components (or widgets), known as the Athena widget set. And the OPEN LOOK Intrinsics Toolkit includes a different set of widgets. Widgets make it easier to create complex applications; they also ensure a consistent user interface between applications. In discussing various clients in this chapter, we'll point out some of the features attributable to the X Toolkit.

For a comprehensive treatment of the X Toolkit, software developers should refer to Volume Four, *X Toolkit Intrinsics Programming Manual*, and Volume Five, *X Toolkit Intrinsics Reference Manual*.

You can start these clients from the command line in any *xterm* window or, if you like, you can add them to your window manager's menu (see Chapter 13, *Customizing olwm*).

You can terminate any of these clients by using the *Quit* button on the *olwm* window menu. As well, some such as *xman*, have their own *Quit* button.

8.1 Desk Accessories

The clients *xclock*, *oclock*, *xcalc*, *xload*, *xbiff*, and *xman* can be thought of as *desk accessories*. "Desk accessories" is a term we've borrowed from the Macintosh environment, meaning small applications available—and useful—at any time. Many of the DeskSet clients described in Chapter 7, *The OpenWindows DeskSet Clients*, can also be considered to be Desk Accessories.

8.1.1 Clock Programs: xclock and oclock

The standard release of X includes two clients that display the time: *xclock* and *oclock* (the *oclock* client was added to the standard distribution of X in Release 4). The time displayed by both *xclock* and *oclock* is the system time set and displayed with the UNIX *date* (1) command, the MS-DOS *date* and *time* commands, etc., in other words, the operating system's notion of the correct time.

xclock continuously displays the time, either in analog or digital form, in a standard window. The analog *xclock* shows a round 12-hour clock face with tick marks representing the minutes. The digital *xclock* shows the 24-hour time (2:30 PM would be represented as 14:30) as well as the day, month, and year. You can run more than one clock at a time. The

analog clock is the default. Figure 8-1 shows two *xclock* applications being run: an analog clock above a digital clock.

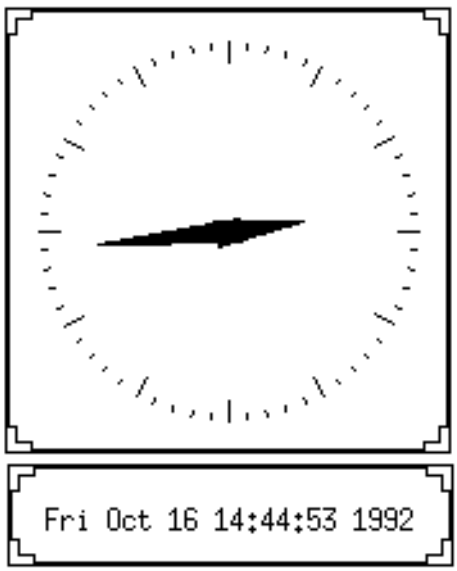


Figure 8-1. Two xclock displays: analog clock above digital clock

Note that *olwm* normally puts a titlebar on all windows. We suppressed this, to show how the *xclock* window would normally look, by adding them to the `MinimalDecor` list. For example, if you normally use *xclock*, you could add this to your *.Xdefaults* file:

```
OpenWindows.MinimalDecor: xclock
```

You don't have to do this, but it makes the *xclock* window look nicer if you do.

The *oclock* client (available as of X11 Release 4) displays the time in analog form on a round 12-hour clock face without tick marks. The only features of an *oclock* display are the round clock outline, hour and minute hands, and the "jewel" marking 12 o'clock.

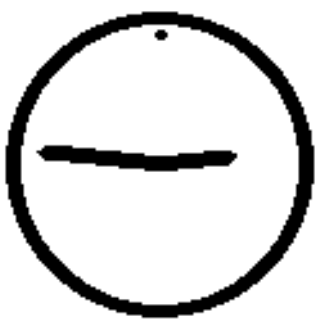


Figure 8-2. oclock display

Though it is somewhat more difficult to read the precise time on the simple *oclock* display, the *oclock* is perhaps a little more aesthetically pleasing than the analog *xclock*.

oclock also makes use of the X Shape server extension, which supports non-rectangular windows. If you try to resize the round *oclock*, you'll discover that it's possible to "stretch" it into various oblong shapes, as shown in Figure 8-3.

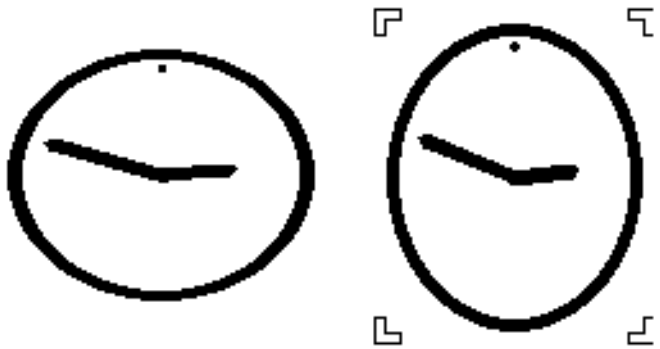


Figure 8-3. Oblong oclock displays

oclock sets some "window manager hints" to tell *olwm* that it doesn't need a titlebar, you don't need to add it to the `MinimalDecor` list. To move the *oclock* display under *olwm*, you need only grab it with the pointer. Click and hold SELECT, the left pointer button, and drag the clock to where you want it. To Resize it, however, is a two-step process:

- Click SELECT on the window's border, and the normal OPEN LOOK resize corners will appear, as in the right side of Figure 8-3.
- Then use these re-size corners as you would on any window.

This is how the two oblong clocks were produced.

Though the default colors for *oclock* are black and white, it was designed to be run in color. The minute hand, hour hand, jewel, clock border, and background can all be set to a color, using either command line options (described in Chapter 11, *Command-line Options*) or by specifying client resources (described in Chapter 12, *Setting Resources*). See the *oclock* reference page in Part Three of this guide for the necessary command line options and color suggestions.

8.1.2 A Scientific Calculator: xcalc

xcalc is a scientific calculator that can emulate a Texas Instruments TI-30 or a Hewlett Packard HP-10C.

OpenWindows users should also check out the operation of *calctool* described in Chapter 7, *The OpenWindows DeskSet Clients*.

Once you place the pointer within the *xcalc* window, the calculator can be operated in two ways: with the pointer, by clicking the first pointer button on the buttons in the calculator window, or with the keyboard, by typing the same numbers and symbols that are displayed in the calculator window. When using the first method, notice that the pointer appears as a small hand, enabling you to “press” the buttons.Figure 8-4 shows *xcalc* on the screen.

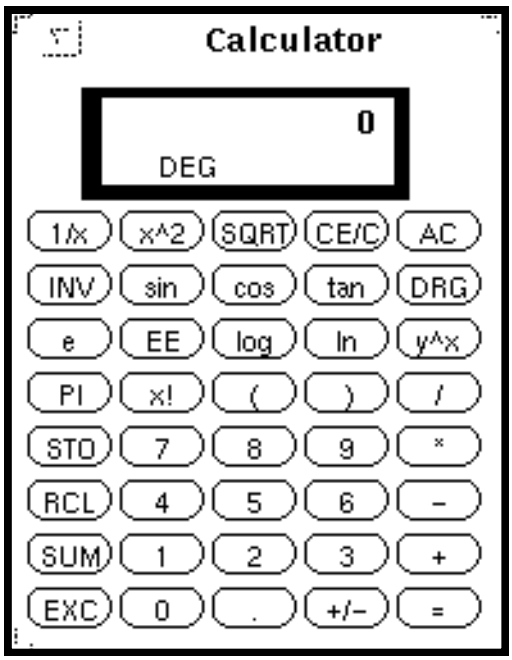


Figure 8-4. The default xcalc (TI-30 mode) on the screen

Figure 8-4 depicts the version of the calculator provided with Release 4 of X. As you can see, it features oval buttons. If you are running an earlier release, the calculator will have rectangular buttons and may also have a darker background color. We are discussing the Release 4 version. For additional information, see the *xcalc* reference page in Part Three of this guide.

The values punched on the calculator and the results of calculations are displayed in the long horizontal window along the top of the *xcalc*. You can enter values either by clicking on the calculator keys with the pointer, or by pressing equivalent keys on the keyboard. Most of the calculator keys have keyboard equivalents. The not-so-obvious equivalents are described on the *xcalc* reference page in Part Three of this guide.

By default, *xcalc* works like a TI-30 calculator. To run *xcalc* in this mode, enter:

```
% xcalc &
```

You can also operate the calculator in Reverse Polish Notation (as an HP-10C calculator operates), by entering:

```
% xcalc -rpn &
```

In Reverse Polish Notation the operands are entered first, then the operator. For example, `5 * 4 =` would be entered as `5 Enter 4 *`. This entry sequence is designed to minimize keystrokes for complex calculations.

xcalc allows you to select the number in the calculator display. You select the number using the first pointer button and paste it in another window using the second button. Since this is an Athena widget set application, you should refer to Appendix A, *The xterm/olterm Terminal Emulator*, for information about copying and pasting text selections.

xcalc can also be resized. In prior releases, this was not possible. Also as of X11R4, the code in *xcalc* to emulate an analog slide rule (`-analog`) is no longer available.

For more information on the function of each of the calculator keys, see the *xcalc* reference page in Part Three of this guide.

You can terminate the calculator by using the *Quit* menu button on the *olwm* window menu. For a method more in keeping with the spirit of this program, you can also terminate it by simulating "powering off" the calculator by clicking the right pointer button on the TI calculator's AC key or the HP calculator's ON key. You can even terminate it by positioning the pointer on the calculator and typing q, Q, or Control-C.

8.1.3 Mail Notification Client: xbiff

xbiff is a simple program that notifies you when you have mail. It puts up a window showing a picture of a mailbox. When you receive new mail, the keyboard emits a beep, the flag on the mailbox goes up, and the image changes to reverse video. Figure 8-5 shows the *xbiff* mailbox before and after mail is received. We didn't need to add *xbiff* to the `MinimalDecor` resource because its program sets certain "window manager hints" that tell *olwm* not to provide a titlebar.

Figure 8-5. xbiff before and after mail is received

After you read your mail (and the mail file is empty), the image changes back to its original state; or you can click on the full mailbox icon with any pointer button to change it back to empty. Regardless of the number of mail messages when you do this, *xbiff* remembers the current size of your mail file to be the *empty* size.

This feature of *xbiff* has two implications, one obvious and the other not so obvious: when additional mail arrives (and the mail file becomes larger), *xbiff* notifies you; but when you delete messages from the current mail file (and the file becomes smaller), *xbiff* also notifies you—when you exit the mail program and the file is saved. This can be a little confusing until you get used to it.

While *xbiff* is intended to monitor a mail file, it can actually be set up to watch any file whose size changes using the `-file` option (followed by the name of the file to be monitored). For instance, if you're running a program that produces output intermittently, you can start *xbiff* with `-file` followed by the name of the output file; then *xbiff* will notify you when output is returned. (You can even specify an image other than the mailbox using resource variables—even for a single *xbiff* process.) See the *xbiff* reference page in Part Three of this guide for a list of options and resources. See Chapter 12, *Setting Resources*, for the syntax of resource specifications.

8.1.4 Monitoring System Load Average: xload

By default, *xload* polls the system for the load average at five-second intervals and displays the results in a simple histogram.

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

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

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

```
% xload &
```

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

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

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

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

Figure 8-6 shows the resulting *kansas* display: two *xload* windows—the top window monitoring activity on the local system and the bottom one monitoring activity on the remote system.

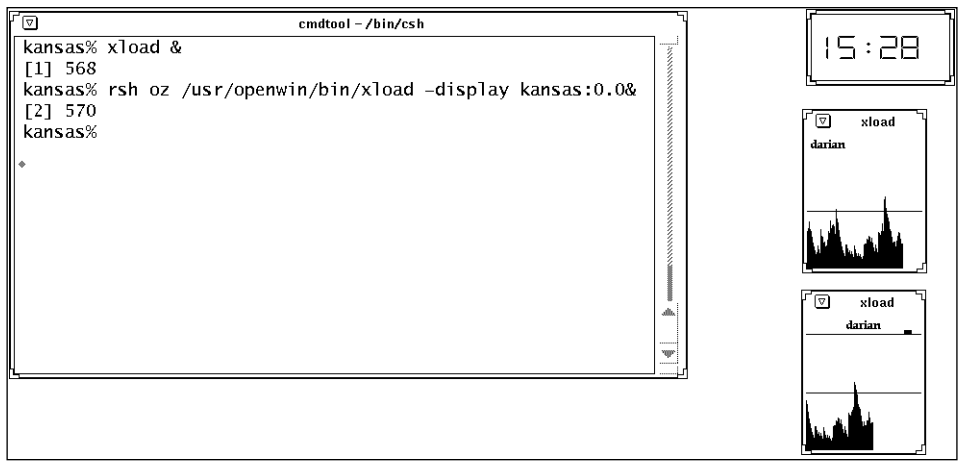


Figure 8-6. Monitoring activity on two systems with xload

For OpenWindows users, a simpler method is to use the more flexible *perfmeter* program, which knows how to reach out across the network to ask another machine for its load average (or any of almost a dozen statistics). *perfmeter* has the advantage that it gives a numerical scale to its graph, rather than the numberless graph that *xload* offers. There is more to *xload*, and much more to *perfmeter*, than these examples show. In particular, *perfmeter* lets you select CPU load, disk I/O, paging, ethernet traffic, and many other things to watch. And you can change the list of hosts you want to watch without quitting and restarting the program. See the description of *perfmeter* in Chapter 7, *The OpenWindows DeskSet Clients*. Also see the reference manual page in Section Three of this guide for more information about either *xload* or *perfmeter*.

8.1.5 Browsing Reference Pages: xman

The *xman* client allows you to display and browse through formatted versions of reference pages. By default, *xman* lets you look at the standard UNIX manual pages found in subdirectories of the directory */usr/man*. The standard version of X assumes there are 10 subdirectories: *man1* through *man8*, corresponding to the eight sections of reference pages in the UNIX documentation set; *manl* (man local) and *mann* (man new). You can specify other directories by setting the MANPATH system variable. (The individual directory names should be separated by colons.)

This section describes the version of *xman* provided with Release 4 of X. From a user's viewpoint, the general operation of the client has not changed much since prior releases but the organization of menus and options has changed.

Regardless of the version of X, you run *xman* by typing:

```
% xman &
```

in an *xterm* window.

The initial *xman* window, shown in Figure 8-7, is a small window containing only a few commands.

Figure 8-7. Initial xman window

This window is small enough to be displayed for prolonged periods during which you might have need to examine UNIX manual pages. You select a command by clicking on it with the first pointer button.

The *Manual Page* command brings up a larger window in which you can display a formatted version of any manual page in the MANPATH. By default, the first page displayed contains general help information about *xman*. Use this information to acquaint yourself with the client's features.* (The actual *xman* reference page in Part Three of this guide primarily describes how to customize the client.)

Once you've opened this larger window, you can display formatted manual pages in it.†

Notice the horizontal bar spanning the top edge of the window. (If you're running *olwm* or a similar window manager, this bar appears beneath the titlebar provided by the window manager.) The bar is divided into three parts labeled *Options*, *Sections*, and *Xman Help*. The part currently labeled *Xman Help* is merely informational and the text displayed in it will change depending on the contents of the window. The parts labeled *Options* and *Sections* are actually handles to two *xman* menus.

If you place the pointer on the *Options* box and press and hold down the first button, a menu called *Xman Options* will be displayed below. The menu is pictured in Figure 8-8.

The functionality of these options is described in the on-line *xman* help page. To select an option, move the pointer down the menu and release the first button on the option you want. The option you will probably use most frequently is the first one, *Display Directory*.

Display Directory lists the reference pages in the current reference page directory (also called a *section*). By default, this is *man1*, the user commands. When you list the contents

† Selecting the **Help** command also opens a large window in which the same help information is displayed. The **Help** command is something of a dead end, however; you cannot display any other text in this window.

Display Directory

Display Manual Page

Help

Search

Show Both Screens

Remove This Manpage

Open New Manpage

Show Version

Quit

Figure 8-8. Xman Options menu

of *man1* in this way, the informational section of the horizontal bar reads ***Directory of: (1) User Commands.***

xman displays each man page directory in a window known as a *viewport*, created with the Athena Viewport widget from the X Toolkit. A viewport is a composite widget that provides a main window and horizontal and/or vertical scrollbars.

xman's scrollbar is also an Athena widget. (See Appendix A, *The xterm/olterm Terminal Emulator*, for instructions on using an Athena scrollbar.) The OPEN LOOK Toolkit also provides a scrollbar, described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, which looks and operates differently than the Athena scrollbar.

For a list of the Athena widgets, see Appendix I, *Athena Widget Resources*. For complete information about the X Toolkit, see Volume Four, *X Toolkit Intrinsics Programming Manual*, and Volume Five, *X Toolkit Intrinsics Reference Manual*.

Once you've listed a reference page directory in the *xman* window, you can display a formatted version of any page in the list simply by clicking on the name with the first pointer button. Figure 8-9 shows the formatted reference page for the UNIX *cd* (1) command.

To display another manual page from the same directory, display the *Xman Options* menu again. Select *Display Directory* and the directory listing is again displayed in the window. Then click on another command name to display its manual page in the window. (If you decide not to display another reference page, you can remove the directory listing and go back to the reference page previously displayed by using the second *Xman Options* menu selection, *Display Manual Page*. *Display Directory* and *Display Manual Page* are toggles of one another.)

To display a manual page from another directory in the MANPATH, you must first change to that directory using the second *xman* menu, *Xman Sections*. Bring up the menu by placing the pointer in the *Sections* box in the application's titlebar and holding down the first button. The *Xman Sections* menu lists the default directories of UNIX manual pages (in this case for a Solaris 2 system), as shown in Figure 8-10.

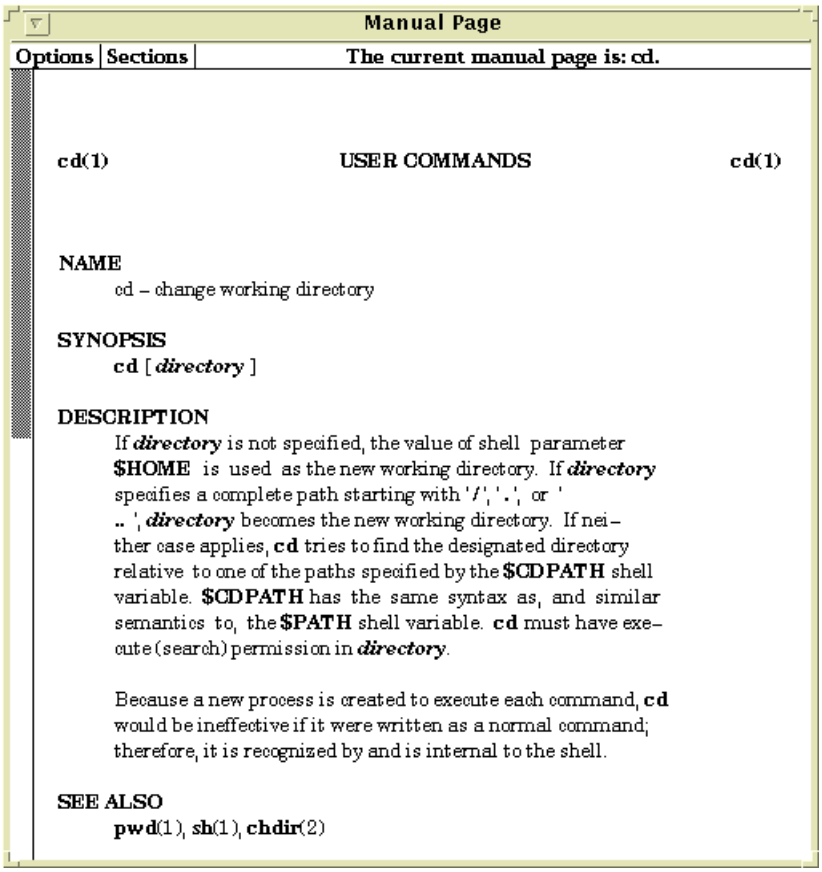


Figure 8-9. cd reference page displayed in xman window

Click on the first pointer button to select another directory of reference pages from which to choose. Once you select a directory, the files in that directory are listed in the window. Again, you display a page by clicking on its name with the first pointer button.

You can display more than one “browsing” window simultaneously by selecting the *Open New Manpage* option from the *Xman Options* menu. An additional reference page window will be opened again starting with the help information.

The various windows *xman* creates can all be iconified and each is represented by a different icon symbol. The icon symbols for the initial *xman* window, the help window, and the browsing window appear in Figure 8-11. Keep in mind that if you’ve displayed several browsing windows simultaneously, you can iconify each of them.

Note that the “Manual Page” icon label is too big to fit in the icon area provide by *olwm*. This is amusing but harmless.

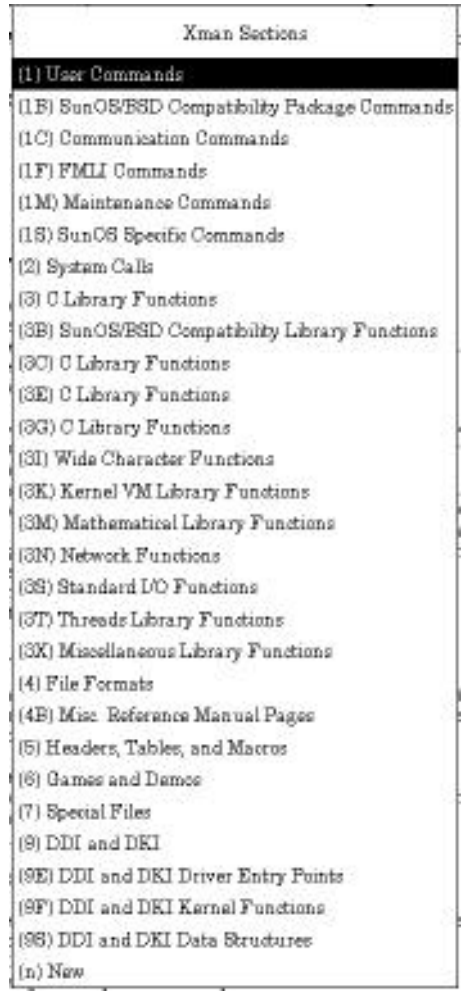


Figure 8-10. Xman Sections menu

Figure 8-11. Icons for xman's initial window, help window, and browsing window

You can remove a browsing window by selecting the *Remove This Manpage* option from the *Xman Options* menu. Selecting *Quit* from the *Xman Options* menu or from the initial *xman* window causes the client to exit.

An alternate man page browser is *tkman*, which is included on the CD-ROM accompanying this book. The Tk toolkit provides a Motif-like interface, but *tkman* is a very useful browser - it can treat almost any text as a hyperlink, and jump to the appropriate man page.

OpenWindows users should also refer to *answerbook* and *helpviewer*, described in Chapter 7, *The OpenWindows DeskSet Clients*.

8.2 The xedit Text Editor

The *xedit* client provides a window in which you can create and edit text files. The editing commands *xedit* recognizes are provided by the Athena Text widget. Many other standard and user-contributed clients also include areas in which you can enter text. Several of these clients, including *xclipboard* and *xmh*, also use the Text widget, and thus recognize the same editing commands as *xedit*.

xedit is in some ways patterned after *textedit*, a version of which has been in SunOS for nearly a decade. OpenWindows users may wish to use *textedit*, described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, rather than *xedit*, since *textedit* uses the OPEN LOOK conventions..

The *xedit* client is valuable to illustrate the use of the Athena Text widget. (*xedit* can also be used to illustrate several other widgets.) However, we do not recommend using *xedit* as your primary text editor. The program's behavior can be erratic. For example, it's fairly easy to overwrite files inadvertently, as explained in the discussion of the *Load* button later in this section. The redraw command (Control-L) causes text in the window to scroll so as to reposition the cursor in the center of the editing window—not a welcome surprise. Some of the commands to create a new paragraph may also inadvertently copy preceding text. These are just a few of *xedit*'s inconvenient features. OpenWindows users, as mentioned, may wish to use *textedit*. A Motif-based free-software editor of similar power is the *asedit* program, source for which is included in your CD-ROM.

Still, it is necessary to know something about the Athena Text widget in order to be able to enter and edit text in windows provided by many standard X11 clients.

xedit recognizes various Control and Meta keystroke combinations that are bound to a set of commands similar to those provided by the *emacs* text editor.†

In addition, you can use the pointer to move the cursor in the text or to select a portion of text. The *xedit* cursor is a caret symbol (^). A caret cursor appears in each of the three areas that accept text entry. (These areas are described later in this section.) Pressing the first pointer button causes the insertion point (cursor) to move to the location of the pointer. Notice that the cursor always appears between characters, rather than on a character as the

† The commands may be bound to keys different from the defaults described below through the standard X Toolkit key translation mechanisms. See Chapter 12, *Setting Resources*, for more information.

xterm cursor does. Double-clicking the first pointer button selects a word, triple-clicking selects a paragraph, and quadruple-clicking selects everything. After you select text, the selection may be extended in either direction by using the third pointer button.

You can invoke *xedit* by entering:

```
% xedit &
```

Since no filename has been specified, the main section of the *xedit* window is empty, as illustrated by Figure 8-12.

Figure 8-12. xedit window before text file is read in

Notice that the *xedit* window is divided into four parts:

- A commands section, which features three command push buttons (*Quit*, *Save*, and *Load*) and an area to their right in which a filename can be entered.
- A message window, which displays messages from the client and can also be used as a scratch pad.
- The filename display, which shows the name of the file being edited and the read/write permissions for the file.
- The edit window, in which the text of the file is displayed and in which you issue the editing commands.

The *xedit* application uses the Athena VPaned widget (of the X Toolkit), which arranges subwindows one above the other without overlapping. The subwindows are also known as *vertical panes* and the non-overlapping, top-to-bottom arrangement is commonly described as *vertical tiling*.

The individual panes organized by a VPaned widget can be any other type of widget. In the case of *xedit*, for example, the commands section is one pane that contains three command buttons (another widget) and a small window to the right of the buttons (a Text widget) in which a filename can be entered.

Notice the three small black rectangles on the borders between the panes. These features are called *grips* and they serve as handles to allow you to resize the subwindows. When the pointer is positioned on the grip and a button pressed, an arrow is displayed that indicates the direction in which the border between the two windows can be moved. If you move the pointer in the direction of the arrow (while pressing the button), one subwindow will grow while the other will shrink.

You can enter text in three areas of the *xedit* window: the message window, the edit window, and the small window immediately to the right of the command buttons in which you can enter a filename. (Thus all three use a Text widget.) Note that the small filename window to the right of the command buttons is different from the filename display (lower in the *xedit* window). The filename display is simply that—a display of the filename; the window does not support editing.

All three areas that permit editing display the caret text cursor. In order to focus keyboard input to a particular area, the pointer must rest in that area—regardless of whether *olwm* is operating with the default click-to-type focus. (If click-to-type focus is in effect, the *xedit* window must also be selected as the focus window.) This is extremely important to remember. Both the message window and the edit window will display a vertical scrollbar if the text is too large to fit. (Be aware also that a scrollbar technically is not part of the text entry window it borders. If the pointer is resting in a scrollbar, keyboard input will be lost—it will not be directed to the corresponding text area!)

The three push buttons in the commands section have the following functions:

Quit Exits the current editing session and closes the window. If changes have not been saved, *xedit* displays a warning in the message window, and does not exit, thus allowing the user to save the file.

Save Writes the file. If file backups are enabled (using the `enableBackups` resource), *xedit* first stores a copy of the unedited file as filename *.BAK* and then overwrites *filename* with the contents of the edit window. The *filename* used is the text that appears in the area immediately to the right of the *Load* button.

Load Loads the file displayed immediately to the right of the button into the edit window. If a file is currently being displayed and has been modified, a warning message will ask the user to save the changes, or to press *Load* again.

This interface has at least two serious pitfalls. First, if you're working on a file that has unsaved changes and you try to load a second file, it's possible to overwrite the second file. This is how it happens. In order to load a second file, you must enter the name of the file in the area next to the *Load* button; then press *Load*. If you try to load a second file while editing a file with unsaved changes, *xedit* warns you to save or press *Load* again. If you press *Save* the current file will be saved—but as the name to the right, the second file you intended to load.

If backups are not enabled, this action will overwrite the file you wanted to load. If backups are enabled, the first file will be saved under the name of the second file with a *.BAK* extension and the second file will not be overwritten. Because of this potential problem, we recommend that you set the resource `enableBackups` to `on` (and load the resources using *xrdb*) before using *xedit*.

A second problem can occur after you've loaded a file by entering the name in the window next to the *Load* button. Say you've been editing the file for some time, but haven't saved the changes. If you go to save the changes and accidentally double-click on *Load* (not that difficult to do), you'll reload the version of the file before you made the edits. The changes are lost!

Now, after considering some of the possible pitfalls, let's load a file into the empty *xedit* window as shown in the figure above. (Obviously, in this case, there's no danger of overwriting an existing file.) To load a file called *index.html*:

1. Place the pointer in the area to the right of the `Load` button.
2. Type *index.html*. The caret cursor moves as you type.
3. Place the pointer on the *Load* command button and press the first pointer button.

The file called *index.html* is displayed in the edit window, as shown in Figure 8-13.

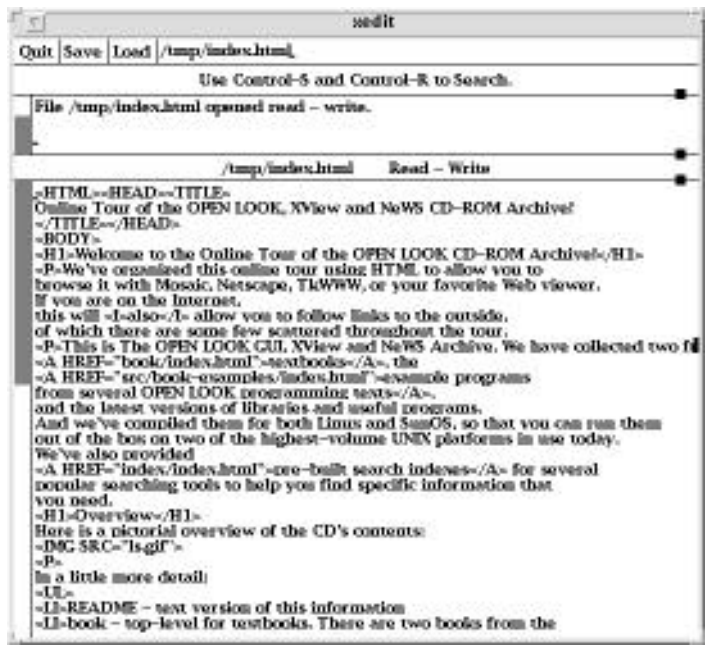


Figure 8-13. text file displayed in xedit window

The simpler commands to edit or append text are intuitive. A backspace deletes the character to the left of the cursor. Typing enters characters immediately before the cursor point, causing the cursor to advance to the right. When you first load a file, the cursor appears at the beginning of the text in the edit window. If you want to append text to the end of the file, move the pointer to the end of the text and click the first button. The caret cursor

appears where the pointer is and any text you type is added to the end of the file, moving the cursor to the right.

The list at the end of this section summarizes all of the editing command recognized by *xedit*. In this list of commands, a *line* refers to one row of characters displayed in the window. A *paragraph* refers to the text between manually inserted carriage returns or blank lines. Text within a paragraph is automatically broken into lines based on the current width of the window.

The keystroke combinations are defined as indicated. (Note that "Control" and "Meta" are two of the "soft" keynames X recognizes. They are mapped to particular physical keys which may vary from keyboard to keyboard. See the "xmodmap" section in Chapter 14, *Customization Clients*, for a discussion of modifier key mapping.) If you are using an earlier release of X, a few of these keystroke combinations may produce slightly different results.

Keep in mind that you can redefine any of these key combinations using what are known as *translations*. Translations allow you to assign actions recognized by a client to particular key combinations, or key and pointer button combinations. For example, *xedit* recognizes actions to delete text, to copy text, to move the cursor, etc. *xedit* defines key combinations to invoke these actions. (The key/action mappings appear in the list at the end of this section.) For information on specifying alternate mappings, see "Event Translations" in Chapter 12, *Setting Resources*.

Note that the function assigned to the Return key in the following list applies only to the edit window and message window. In the filename window (next to the command buttons), Return simply moves the cursor to the end of the line.

Table 8-1. Xedit key bindings

Key sequence	Function
Control-A	Move to the beginning of the current line.
Control-B	Move backward one character.
Control-D	Delete the next character.
Control-E	Move to the end of the current line.
Control-F	Move forward one character.
Control-H or Backspace	Delete the previous character.
Return, Control-J, Control-M, or LineFeed	New paragraph. (**Linefeed**, **Control-J**, and **Control-M** may be unreliable on some terminals.)

Table 8-1. Xedit key bindings

Key sequence	Function
Control-K	Kill the rest of this line. (Does not kill the carriage return at the end of the line. To do so, use **Control-K** twice. However, be aware that the second kill overwrites the text line in the kill buffer.)
Control-L	Redraw the window. (Also scrolls text so that cursor is positioned in the middle of the window.)
Control-N	Move down to the next line.
Control-O	Divide this line into two lines at this point.
Control-P	Move up to the previous line.
Control-V	Move down to the next screenful of text.
Control-W	Kill the selected text.
Control-Y	Insert the last killed text. (If the last killed text is a carriage return—see Control-K above—a blank line is inserted.)
Control-Z	Scroll up the text one line.
Meta-<	Move to the beginning of the file.
Meta->	Move to the end of the file.
Meta-[	Move backward one paragraph.
Meta-]	Move forward one paragraph.
Meta-B	Move backward one word.
Meta-D	Kill the next word.
Meta-F	Move forward one word.
Meta-H or Meta-Delete	Kill the previous word.
Meta-I	Insert a file. If any text is selected, use the selected text as the filename. Otherwise, a dialog box will appear in which you can type the desired filename.
Meta-V	Move up to the previous screenful of text.

Table 8-1. Xedit key bindings

Key sequence	Function
Meta-Y	Insert the last selected text here. Note that this can be text selected in some other text subwindow. Also, if you select some text in an *xterm* window, it may be inserted in an *xmh* window with this command. Pressing pointer button 2 is equivalent to this command.
Meta-Z	Scroll down the text one line.
Delete	Delete the previous character.

8.3 Window and Display Information Clients

The standard release of X includes four clients that provide information about windows on the display and about the display itself. Much of the information is probably more relevant to a programmer than to the typical user. However, these clients also provide certain pieces of information, such as window geometry, window ID numbers, and the number and nature of screens on the display, that can assist you in using other clients.

8.3.1 Displaying Information about a Window: xwininfo

The *xwininfo* client displays information about a particular window. Some of this information can be useful in determining or setting window geometry (described in Chapter 3). *xwininfo* also provides you with the *window ID* (also called the resource ID). Each window has a unique identification number associated with it. This number can be used as a command line argument with several clients. Most notably, the window ID can be supplied to the *xkill* client to specify the window be killed.

You can also use the window ID as an argument to the *xprop* client, which displays various window properties. As described in Chapter 1, a property is a piece of information associated with a window or a font and stored in the server. Properties facilitate interclient communication; they are used by clients to store information that other clients might need to know. Storing properties in the server makes the information they contain accessible to all clients. See Chapter 1, the *xprop* reference page in Part Three of this guide, and Volume One, *Xlib Programming Manual*, for more information about properties and the *xprop* client.

To display information about a window, type this command in an *xterm* window:

```
% xwininfo
```

The pointer changes to the cross-hair pointer and you are directed to select the window about which you want information:

```
xwininfo ==> Please select the window about which you
==> would like information by clicking the
```

```
    ==> mouse in that window.
```

You can select any window on the display, including the window in which you've typed the command and the root window. (Rather than using the pointer, you can specify a window on the command line by supplying its title, or name if it has no title, as an argument to *xwininfo*'s own `-name` option. See Chapter 9 for information about setting a client's title and name. See the *xwininfo* reference page in Part Three of this guide for a list of its options.)

The following diagram, Figure 8-14, shows the statistics of *xwininfo* supplies with some typical readings.

```
xwininfo ==> Window id: 0x70000e (xterm)
==> Absolute upper-left X: 12
==> Absolute upper-left Y: 29
==> Relative upper-left X: 0
==> Relative upper-left Y: 0
==> Width: 818 ==> Height: 484
==> Depth: 8 ==> Border width: 0
==> Window class: InputOutput
==> Colormap: 0x8006b (installed)
==> Window Bit Gravity State: NorthWestGravity
==> Window Window Gravity State: NorthWestGravity
==> Window Backing Store State: NotUseful
==> Window Save Under State: no
==> Window Map State: IsViewable
==> Window Override Redirect State: no
==> Corners: +12+29 -322+29 -322-387 +12-387
```

Figure 8-14. Window information displayed by xwininfo

These readings are for a login *xterm* window displayed using a 12-point Roman Courier font. All numerical information is in pixels, except depth, which is in bits per pixel. The *olwm* window manager is also running. The most significant statistics from the above figure for the average user are:

```
xwininfo ==> Window id: 0x70000e (xterm)
==> Absolute upper-left X: 12
==> Absolute upper-left Y: 29
==> Relative upper-left X: 0
==> Relative upper-left Y: 0
==> Width: 818 ==> Height: 484
==> Depth: 8 ==> Border width: 0
==> Colormap: 0x8006b (installed)
==> Corners: +12+29 -322+29 -322-387 +12-387
```

The first piece of information is the window ID, which can be used as an argument to *xkill*. Specifying the window to be killed by its ID number is somewhat less risky than choosing it with the pointer.

With many window managers, you can use some of the other statistics to gauge the window's geometry (size and position). Generally, the absolute upper-left X and Y correspond to the positive x and y offsets that can be supplied to the `-geometry` option used to place

a client window. (The use of the `-geometry` option is discussed in Chapter 3, *Opening Additional Windows*.).

The Window Manager window frame complicates matters. When *olwm* or another framing window manager such as *mwm* is running, the absolute upper-left X and Y correspond to the x and y coordinates of the application window—but not the framed window!

Let's take another look at the sample *xwininfo* output. The absolute upper-left X and Y suggest that the window is located at coordinates 12,29. However, the output is actually for an *xterm* located at coordinates 0,0! The 12,29 are the coordinates of the *xterm* window itself; the coordinates represent the distance of the window from 0,0 *including the dimensions of the frame.* The default frame for *mwm* is actually 12 pixels in the x dimension and 29 pixels in the y dimension (because of the titlebar).

For any given window manager and set of font and size resources, there will be some fixed offset (like the 12,29 above) that you can subtract to get the actual position.These figures can be supplied as arguments to the `-geometry` option on the command line to specify window placement, as described in Chapter 3, *Opening Additional Windows*. (Chapter 3 also describes two simpler method of gauging x and y offsets.)

The relative upper-left X and Y may or may not be meaningful depending upon which window manager you are using. With *twm*, for example, regardless of a window's location, the relative upper-left X and Y are 0 and 0.

The four corners (again, including the frame) are listed with the upper-left corner first and the other three clockwise around the window (i.e., upper-right, lower-right, lower-left). The coordinates of the upper-left corner are, of course, the absolute upper-left X and Y. The width and height in pixels are somewhat less useful, since the geometry option to *xterm* requires that these figures be specified in characters and lines.

The values for window depth and colormap relate to how color is specified. See the discussion of color in Chapter 12, *Setting Resources*, for more information.

The other statistics provided by *xwininfo* are listed below:

```
==> Window class: InputOutput
==> Window Bit Gravity State: NorthWestGravity
==> Window Window Gravity State: NorthWestGravity
==> Window Backing Store State: NotUseful
==> Window Save Under State: no
==> Window Map State: IsViewable
==> Window Override Redirect State: no
```

These statistics have to do with the underlying mechanics of how a window is resized, moved, obscured, unobscured, and otherwise manipulated. They are inherent in the client program and you cannot specify alternatives. For more information on these and other window attributes, see Chapter 4 in Volume One, *Xlib Programming Manual*.

You can also use *xwininfo* with various options to display other window attributes. See the reference page in Part Three of this guide for details.

8.3.2 Listing the Window Tree: xlswins

Windows are arranged in a hierarchy, much like a family tree, with the root window at the top. The *xlswins* client displays the window tree starting with the root window, listing each window by its resource ID and title (or name), if it has one. (See Chapter 9 for a discussion of setting a client's title and name with command line options.)

A resource ID can be supplied to *xkill* to specify the window to kill. You can also supply a resource ID to *xwininfo* to specify the window you want information about, or to *xprop* to get the window's properties. Being able to display the ID numbers of all windows on the screen simultaneously is especially helpful if one or more windows is obscured in the stack. The *xwininfo* client is virtually useless in situations in which one window is hidden behind another. *xlswins* allows you to determine by process of elimination which window is hidden—without having to circulate all the windows on your screen. You can then use *xwininfo* with the ID number (displayed by *xlswins*) to get information about the obscured window.

Figure 8-17 shows the results of *xlswins* for a simple window arrangement: a single *xterm* (login) window on a root window. (No window manager is running.)

```
0x8006e ()
 0x30000e (xterm)
 0x300015 ()
 0x300016 ()
```

Figure 8-15. Window tree displayed by xlswins

The *xterm* window is easily identified. Any client that displays an application window, such as *xterm*, *xclock*, *xfd*, *bitmap*, etc., will be listed by name (in parentheses) following the ID number†. The root window is listed above the *xterm* in the window hierarchy. Client (and other) windows displayed on the root window are called *children* of the root window, in keeping with the family tree analogy; thus, the root window is the parent of the *xterm* window. In the *xlswins* listing, a child window is indented once under its parent.

But what are the other windows listed in Figure 8-15? A superficial examination of these other windows provides a brief introduction to the inner workings of X. An underlying feature of X is that menus, boxes, icons, and even *features* of client windows, such as scrollbars, are actually windows in their own right. What's more, these windows (and client window icons) may still exist, even when they are not displayed.

The two remaining windows are unnamed. From the relative indents of the windows, we can tell certain information. The first unnamed window is a child of the *xterm*, the second is a child of the child.

If we again run *xlswins*, this time requesting a long listing (with the `-l` option), we get geometry information that helps identify each window. This is shown in Figure 8-16.

† Most likely, you will not have to deal with the ID numbers for windows other than the explicitly named client windows. You can use the IDs of the client windows in all of the ways we've discussed: with *xkill*, *xwininfo*, *xprop*, etc.

```
0: 0x8006e (); ()() 1152x900+0+0 +0+0
1: 0x30000e (xterm); (xterm)(XTerm) 818x484+0+0 +0+0
2: 0x300015 (); ()() 818x484+0+0 +1+1
3: 0x300016 (); ()() 14x484+-1+-1 +0+0
```

Figure 8-16. Window tree with geometry specifications

The first number on each line refers to the level of the window in the hierarchy, the root window being at level 0, client windows at 1, etc. Following the *xterm* application window are what are known as the instance and class resource names for the client (in this case, `xterm`, `XTerm`). You use the instance and class resource names to specify default window characteristics, generally by placing them in a file in your home directory. This is described in detail in Chapter 12, *Setting Resources*.

The first geometry string is the complete specification relative to the *parent* window. The second geometry string is the current position relative to the *root* window. Since *olwm* is not running, frames are not an issue. Thus, a window at coordinates 0,0 would have the position `+0+0` relative to the root.

The two unnamed windows under *xterm* are the VT102 window and the window's scrollbar, respectively. (The first *xterm* listing is the application shell window, which can be displayed both as a VT102 and a Tektronix window.)

The listing in Figure 8-16 was generated when no window manager was running. If *olwm* is running, the *xlswins* output is considerably more complicated. Many of the features provided by *olwm*, such as the window frame and its command buttons, and the *Root Menu* and *Window Menu*, are actually windows themselves. This *greatly* complicates the window hierarchy. If you run *xlswins* while *olwm* is running, even if the display has only a single application window, the output will be dozens of lines long; you can assume that most of the mysterious windows in the hierarchy are features provided by the window manager. We just ran *xlswins* on a system with one *cmdtool*, one *contool*, and a publishing software package with one document open, another iconified, and three popups pinned up. There were so many windows that we had to use the standard UNIX utility *wc* (word count) to count them all:

```
% xlswins | wc -l
 149
%
```

You may also notice that application windows, such as *xterm*, are now at level 3 in the hierarchy. This is because *olwm reparents* all client windows; that is, the window manager creates another window that is the parent window of the application window and is itself the child of the root window. (The frame is actually a window in its own right; think of the window manager as creating a window that contains the application window.)

The geometry strings for application windows will also be different when *olwm* is running because of this reparenting and because of the presence of the frame. The first geometry string, which gives the position relative to the parent window, will always end with the *x*,*y* coordinates +0+0, since the parent is the window manager. The second geometry string, which gives the position relative to the root window, will include the dimensions of the frame. A window located at coordinates 0,0 will have the string +12,+29 because the x and

y dimensions of the default frame are 12 and 29 pixels, respectively. See the preceding discussion of *xwininfo* for more information.

For more information on the window hierarchy, see Volume One, *Xlib Programming Manual*.

8.3.3 Listing the Currently Running Clients: xlsclients

You can get a listing of the client applications running on a particular display by using *xlsclients*. Without any options, *xlsclients* displays a two-column list, similar to:

```
colorful xterm -geometry 80x24+10+10 -ls
colorful xclock -geometry -0-0
```

The first column shows the name of the display (machine) and the second the client running on it. The client is represented by the command line used to initiate the process.

This sample listing indicates that there is one *xterm* window and one *xclock* window running on the display `colorful`. (The option `-ls` following the *xterm* command reveals that the shell running in this window is a login shell.)

You can use *xlsclients* to create an *.xsession* or *.xinitrc* file, which specifies the clients you want to be run automatically when you log in. In order to do this, you must have set up client windows in an arrangement you like using command line options alone (that is, without having moved or resized windows via the window manager). You can then run *xlsclients* to print a summary of the command lines you used to set up the display and include those command lines in your *.xsession* or *.xinitrc* file. See "Customizing your Session Start-up" on Page 72, for information on configuring your session.

By default, *xlsclients* lists the clients running on the display corresponding to the DISPLAY environment variable, almost always the local display. You can list the clients running on another display by using the `-display` command line option. See Chapter 3, *Opening Additional Windows*, for more information about the `-display` option.

With the option `-l` (indicating long), *xlsclients* generates a more detailed listing. Figure 8-17 shows the long version of the listing on the previous page.

```
   Window 0x30000e:
 Machine: colorful
 Name: xterm
 Icon Name: xterm
 Command: xterm -geometry 80x24+10+10 -ls
 Instance/Class: xterm/XTerm
Window 0x40000b:
 Machine: colorful
 Name: xclock
 Icon Name: xclock
 Command: xclock -geometry -0-0
 Instance/Class: xclock/XClock
```

Figure 8-17. Long xlsclients listing

For each client, *xlsclients* displays six items of information: the window ID number, machine name, client name, icon name, command line used to run the client, and the instance and class resource names associated with the client.

As we'll see in Chapter 9, many clients, including *xterm*, allow you to specify an alternate name for a client and a title for the client's window. If you've specified a title, it will appear in the *xlsclients* `Name` field. If you haven't specified a title but have specified a name for the application, the name will appear in this field. Neither of the clients in the sample display has been given an alternate name or title.

You use the instance and class resource names to specify default window characteristics, generally by placing them in a file in your home directory. This is described in detail in Chapter 10, *Setting Resources*.

8.3.4 Listing the Currently Running OpenWindows Clients: psps

psps is an OpenWindows-specific tool to list the currently-running xnews clients, both those talking the X11 protocol and those using the NeWS protocol. Its output will be useful primarily to those developing NeWS applications, since there is, for example, no documented way to kill a NeWS client by its process id. In particular, the "ID" numbers do *not* correspond to the X11 "resource ID" used with *xkill* described above; trying to use the PS ID as the `-id` argument to *xkill* results in a "BadValue' X Error, indicating that X11 does not recognize the ID as a resource ID.

Here is an example output:

```
% psps
 ID State Pri ESS OSS DSS Name
> 0x2e0088 runnable 0 6 3 2 psps
> 0x2e2218 IO_wait 100 2 0 3 darian X11 client
> 0x2de218 IO_wait 100 2 0 3 darian X11 client
> 0x232218 runnable 100 2 0 3 darian X11 client
> 0x2c2dbc input_wait 0 2 0 2 RoundBaseFrame
> 0x236218 IO_wait 100 2 0 3 darian X11 client
> 0x212218 IO_wait 100 2 0 3 darian X11 client
> 0x19e218 IO_wait 100 2 0 3 darian X11 client
> 0x204218 IO_wait 100 2 0 3 darian X11 client
> 0x206218 IO_wait 100 2 0 3 darian X11 client
> 0x1f4218 IO_wait 100 2 0 3 darian X11 client
> 0x1f6218 IO_wait 100 2 0 3 darian X11 client
> 0x1ed440 IO_wait 100 4 3 1 X Listener %socketl/tmp/.X11-unix/X0
> 0x1edb5c IO_wait 100 4 3 1 X Listener %socketl6000
> 0x1e21e0 IO_wait 100 4 3 1 NeWS Listener unix
> 0x1e0c54 IO_wait 100 4 3 1 NeWS Listener tcp
> 0x1d638c input_wait 100 6 1 2 Global System EventMgr
> 0x1d79f8 input_wait 0 6 1 2 Global UI EventMgr
> 0x1be500 zombie 0 0 0 0 nullprocess
> 0x199e88 zombie 0 0 0 0 NullCanvasEventMgr
%
```

As you can see, the output resembles a UNIX *ps* command output, hence the name *psps* (PostScript Process Status). It is in fact a listing of the "lightweight processes" running inside the X/NeWS server program.

The last column is a textual description of the program; X11 clients are listed by the host they are from, but are not further identified. NeWS clients are identified by the type of their top-level window; *RoundBaseFrame* is the round NeWS clock shown in several of the examples in this book. Details of the other columns may be found in the reference page in Section Three of this guide.

8.3.5 Getting Information about the Display: xdpyinfo

The *xdpyinfo* client gives information about the X display, including the name of the display (contents of the DISPLAY variable), version and release of X, number of screens, current screen, and statistics relating to the color, resolution, input, and storage capabilities of each screen. The *xdpyinfo* reference page in Part Three of this guide shows a listing for a display that supports both a color and monochrome screen.

Much of the information provided by *xdpyinfo* has to do with how clients communicate information to one another and is more relevant to a programmer than to the typical user. However, the basic statistics about the name of the display, the version and release of X, and the number and nature of screens might be very helpful to a user, particularly one who is using a display for the first time.

The output of *xdpyinfo* appears on its "standard output", normally it appears in the terminal emulator window from which you started it. The output is voluminous; you probably want to pipe it through *more* or *pg*.

In addition, the detailed information about each screen's color capabilities can also be very valuable in learning how to use color more effectively. This information includes the default number of colormap cells: the number of colors you can use on the display at any one time. See Chapter 12, *Setting Resources*, for more information on the use of color and how to specify colors for many clients.

If you are of a programmatic bent, refer to Volume One, *Xlib Programming Manual*, for insights into some of the other information provided by *xdpyinfo*.

8.4 Killing a Client Window

You can normally kill any client with the *Quit* item at the bottom of the OPEN LOOK Window Manager's *Window* menu. For this reason, most OPEN LOOK clients do not provide a special *Quit* button or command. Programs such as terminal emulators, of course, can be terminated by exiting the shell in them, either by the *exit* keyword or by typing your EOF character (normally CTRL/D). However, there are a few cases when you need something more drastic; these should only be used as a last resort, as they *are* drastic. *xkill* kills an X client, and *pam* hides (but doesn't kill off) a NeWS window. These are described in the following sections.

Generally, you should exhaust the safer alternatives before you use *xkill* and other commands that kill a client. When you want to remove a window, depending on the client and what commands it recognizes, try these methods (roughly) in this order:

1. The *Quit* item on the window menu
2. Methods that cause the client to exit after finishing relevant processes:

- Special commands (e.g., *exit*) or key sequences (e.g., Control-D, Control-C, q, Q) recommended to stop a client.
- Certain application-specific menu items (e.g., for *xterm*, the **Main Options** menu commands **Send HUP Signal**, **Send TERM Signal**, and **Quit**).
- The *Quit* button on those applications which provide one

3. When these methods don't work, *then* use commands or menu items that kill the client:

- The **Send KILL Signal** item on the *xterm* **Main Options** menu, for removing *xterm* windows only. (See Appendix A, *The xterm/olterm Terminal Emulator*.)
- The UNIX *kill* command with the client's process ID number, which is determined using *ps*. (This method of removing a window is described for *xclock* earlier in this chapter.*)†
- The *xkill* client (described below) or, for NeWS clients, *pam* (see Section 8.4.2, "Killing a window with pam NeWS-based (OpenWindows only).

8.4.1 Killing a Client with xkill

The *xkill* program allows you to kill a client window or, more specifically, to force the server to end the connection to the client. The process exits and the associated window is removed.

xkill is a fairly drastic method of terminating a client and should *not* be used as the method of choice. In most cases, clients can be terminated in other ways. The possible repercussions of using *xkill* and some of the alternatives are discussed in the next section.

xkill is intended primarily to be used in cases where more conventional methods of removing a client window do not work. It is especially useful when programs have displayed undesired windows on the screen. To remove a stubborn client window, type:

```
% xkill
```

on the command line of an *xterm* window. The pointer changes to a "draped box" pointer and you are instructed to:

```
Select the window whose client you wish to kill with button 1 . . .
```

† This method is powerful but in practice has limitations. UNIX only allows you to kill a process if you are the owner of the process or if you are root. Thus, if a client has been started on your display from a remote system and you don't know the root password, you may not be in a position to use the UNIX *kill* command.

Move the draped box pointer to the window you want to remove, as shown in Figure 8-18, and click the first pointer button. The window is removed. (*xkill* does not allow you to select the root window.)

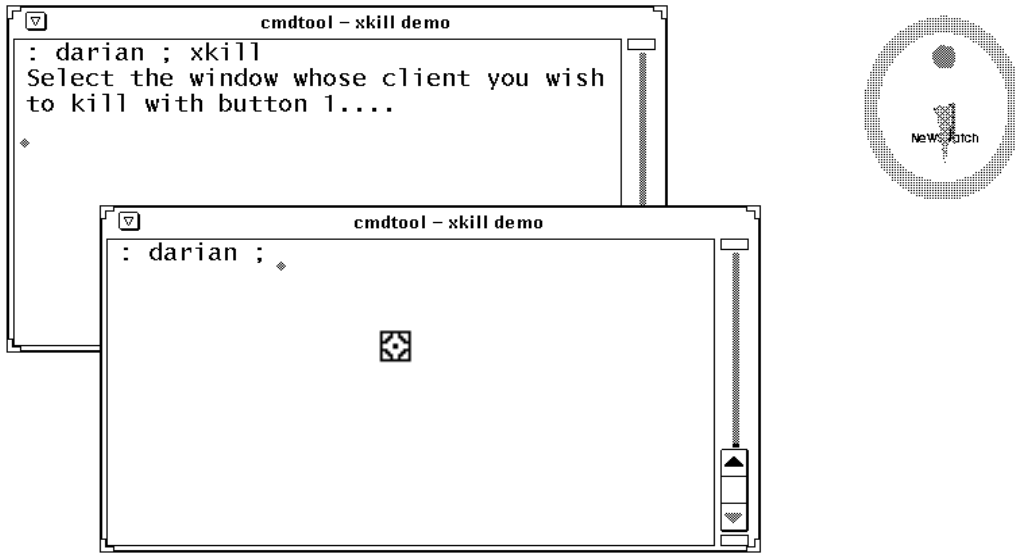


Figure 8-18. Selecting the window to be removed.

You can also specify the window to be killed by its *window ID* (also called the *resource ID*). Every window has an identification number associated with it. The *xwininfo* client can be used to display a window's window/resource ID (see the section "Window and Display Information" later in this chapter).

To remove a window using its ID number, type:

```
% xkill -id number
```

The window with the ID ***number*** is removed. Killing a window by its ID number is more cumbersome but it's somewhat safer than choosing the window to be killed with the pointer. It's too easy to click in the wrong place. (Of course, it's less treacherous to use the pointer on an isolated window than a window in a stack.)

Do not use *xkill* on an icon window, as they are always created by the window manager, not the application; if you do use xkill on the icon, therefore, you will kill your window manager and not the application!

8.4.2 Killing a window with pam NeWS-based (OpenWindows only)

It sometimes happens that a NeWS client will die and leave its window up on the screen. To "hide" windows permanently, there is an OpenWindows tool called "pam".

It works like *xkill* in that it gives you a special cursor that you click on a window, and gives you the message (on standard output):

```
Click on the stuck window to spray some Pam (tm) on it...
```

When it has hidden the window, it says:

```
Unstuck! Remember, you can't see it, but it's still there.
```

to remind you that any resources the window was consuming are still in use. Further, if you use *pam* on an X window that is active, the window is simply unmapped; the process is *not* killed; you would then have to use the UNIX *kill* command to terminate it, as there is no good way to re-map a *pam*med X11 window.

8.5 Demonstration Programs and Games

In one sense, many of the standard MIT clients are "demonstration" programs as they are intended to demonstrate the Athena widgets, both to users and to programmers. But some are specifically written to demonstrate aspects of The X Window System, and others are games for the pure and simple fun of playing.

Table 8-2. X11 Demonstrations and Games

Name	Function	Notes
plaid		
puzzle		
MORE	TO	COME

We've now looked at most of the standard clients in a distribution of The X Window System. Now we turn our attention to some special-purpose graphics clients, both in standard X and in the OPEN LOOK distributions.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 9

Graphics Clients

The X Window System is rich in graphical capabilities†. This chapter explores some of the graphics utility programs that come standard with X, as well as some of the OPEN LOOK extensions and some commercial and contributed software. We look at:

- programs for collecting and viewing bitmapped graphics from the system,
- programs for editing bitmap files,
- a brief survey of commercial desktop graphics programs for the OPEN LOOK GUI,
- programs for displaying PostScript graphical images, and
- programs for editing font images for use in X11.

9.1 Bitmap Gathering and Viewing

A *bitmap* or *raster* is a grid of pixels, or picture elements, each of which is white, black, or in the case of a color display, a color. There are several methods of obtaining bitmap files. This section describes several programs that use a method called “screen capture”; this term means recording in a file the contents of the actual display. Another is “bitmap editing”, discussed in Section 9.2, “Bitmap Editing and Conversion” on Page 231.

After you’ve been using any window system for a while, there comes a time when you need to record the image of what is on the screen. It may be that you are having problems with somebody else’s software, and want to show exactly what all the pop-up windows looked like at a particular time. Or you may be writing documentation about how to use some par-

† For *developing* full-fledged graphics software, see PHIGS, the Programmers’ Hierarchical Interactive Graphics Systems, and PEX, the Phigs Extension to X, described in the *PHIGS Programming Manual*. When PHIGS/PEX becomes more popular, we can expect to see a real increase in the number of impressive graphics programs running under X.

ticular graphics tool, and want to get a bitmap file to show how the screen looks at some particular point. OpenWindows features a program called *snapshot* that can grab the whole screen, one window, or a defined region, and lets you save or print the image. *xwd* is a standard X11 program to dump an entire window; it can be used with some filter programs to print screen windows. And all X11 distributions include a program called *xmag* for looking at part of the screen.

Two contributed clients, *xloadimage* and *xv*, do a good job of displaying graphics. They specialize in different jobs: *xloadimage* can load well-nigh any graphics image you are likely to find. The *xv* client can load a smaller variety of image formats, but can do far more with the image once it's loaded, including cropping, scaling, and writing it out. Both programs have the convenient property that they can directly read files compressed with the standard UNIX *compress* facility. *xv* can also grab and save a region of the screen.

See also the unbundled Sun *ShowMe* program in Section 7.18, "ShowMe - graphical conferencing" on Page 180.

9.1.1 Snapshot – the on-screen photographer (OpenWindows)

As we've seen, you can use *xwd* to dump one window or the whole screen, but it's cumbersome to use when you want a region of the screen that doesn't correspond to one window. That is where *snapshot* comes in. *Snapshot* takes a picture of any part of your screen, just as if you had taken an instant camera and shot a photograph of the screen. The advantages of *snapshot* are that it's always in focus, never runs out of film (until your disk is full), and lets you record either the whole screen, or any part of it, or any one specific window. You'd record the whole screen only rarely. To record one window with some dialog boxes, you might use the region (part of screen) mode. And to show a close-up of one program's window, just record that window.

Figure 9-1 shows the control panel for snapshot†.

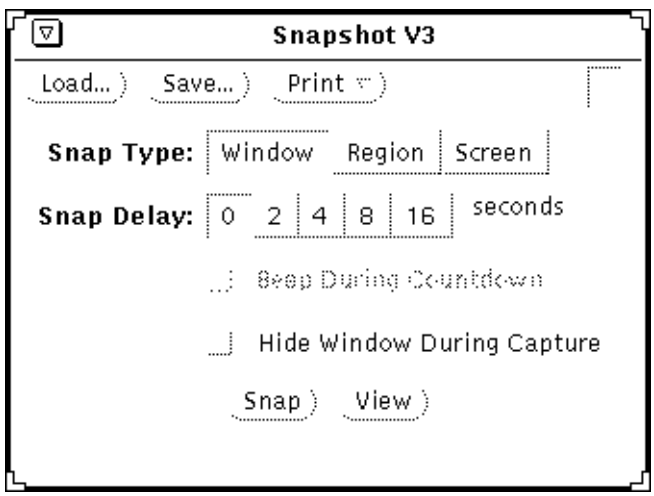


Figure 9-1. Snapshot's control panel

The main operations are `Snap` and `View`. The three kinds of things you can "snap" are a single `Window`, a selected `Region`, or the entire `Screen`. Once you have a snapshot in the program's memory, you can select `View` to display it (to ensure that what you saw is what you got, or to view a snapshot that you loaded with `Load`), or `Print`. And you can `Load` or `Save` an image; the former lets you use this program as a viewer as well as a snapshot program. Let's look at each of these operations in a bit more detail now.

When you press `Snap` with the Type set to `Window`, the `Snap` button becomes busy. Here we are about to take a picture:

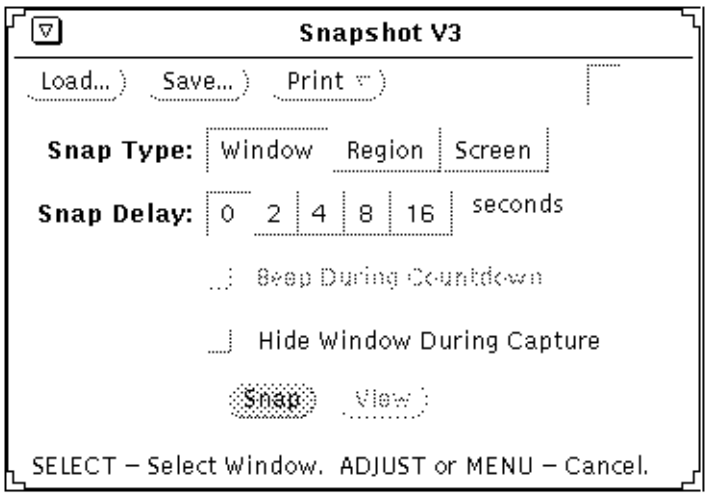


Figure 9-2. Snapshot: Snap Window operation

Notice that the left footer becomes

```
SELECT - Select Window. ADJUST or MENU - Cancel.
```

to tell you what your two choices are. To snapshot the window you want, then, move the pointer into that window, and click the SELECT mouse button. Now *snapshot* goes and reads the contents of the window from X, and saves it for later writing into a file. When you select a single window, snapshot gives you the image of just that window - the titlebar is excluded. To include the titlebar, simply to click the button in the *olwm* titlebar instead of in the window.Your normal next step is *View* to convince yourself that you really succeeded in saving the given window into the program's buffer. Just press the SELECT mouse button on *View* and you will get a display that should be a copy of the window you clicked in. When done with this display, un-pin it or pull down its *Window Menu* and release on *Dismiss*. Then `Save` the image to a file if it's what you want. You can save the file either by the *Save...* menu button, or by dragging it from the drag-and-drop target onto any running File Manager, which will save it under the name *raster* (or *raster0*, *raster1*, ...)

† The OpenWindows Version 2 snapshot has a different window layout but behaves in essentially the same way.

in its current directory; this may be preferable to trying to type a long directory path in the *Save* dialog.

The `Snap Region` control works similarly, but lets you select *any* rectangular area of the screen. For example, if we want to document how a row of icons looks:

Figure 9-3. The Desired Picture

We can move the pointer to one corner (say the upper left), click SELECT and hold it, drag the pointer to the opposite corner (in this case the lower right), and let go. Then when we click the ADJUST button, the snapshot will actually be taken. If we realize we didn't want to proceed, we just click the MENU button to cancel the operation (the word Cancel sometimes got truncated due to a bug in Version 2, or even in Version 3 with -scale large in effect).

If you want to capture an image of the *entire* screen, set the Type to Screen and click SELECT, and it will be done. There is no request for you to Click SELECT, since there is no need; there only is one root window or screen,† so selecting *Snap Screen* is enough – *snapshot* goes ahead and does it. Immediately. This may not always be what you want, though. If you are trying to arrange the entire display, but you don't want the camera (*snapshot*) to appear in the picture, you can check the *Hide Snapshot During Capture* checkbox. When you select this, it automatically sets the timer to 8 seconds, and keeps it there as long as *Hide Snapshot During Capture* is enabled. When not using the Hide feature, the timer can be set to any number of seconds shown in the exclusive choice - 0, 2, 4, 8 or 16. And an audible countdown can be enabled or disabled by selecting the *Timer Bell* toggle. The timer will normally beep while it is running.

If you try to take a snapshot of a program that has a popped up menu or a notice, you will notice that it doesn't work. This is because popup menus and notices use what is called a *grab* in X11 terminology – they take exclusive control of the X Server, and mouse button clicks anywhere in the screen are passed to that program. What you need is a way to make the program run without doing the grab. You have to re-start the application in a special non-grab mode, popup the menu or notice that you want to show, and then you usually have to kill off the application, because it cannot work without the grab. If the program was written with the XView toolkit, use the `-Wfsdb` (window full-screen debug) command line option. For example, Figure 9-2 was produced with the command

```
snapshot -Wfsdb &
```

† If you're on a machine with multiple screens, the *Snap Screen* option captures the screen that it is running on.

to make the *snapshot* program put up its notifier without grabbing the server; this allowed me to click on *snapshot*'s `Snap Region` command and have the mouse clicks passed to *snapshot* rather than just being able to click in the notice.

Another method of grabbing such things is to use the delay setting along with `Snap Region`, and pop up the menu during the delay.

The control *View Snap* has already been discussed; it displays the image in the program's buffer. You can even use this to display Sun Raster format files created by other programs: as long as the name in the *Directory* and *File* text fields matches a valid file, *snapshot* will try to display it.

`Load...` and `Save...` let you save your snapshot, or load another one for examination. In either case, a simple pinnable Load/Save dialog box is popped up in which you can specify the directory (if not the current directory) and the name of the file.

Lastly, the *Print* control lets you print the current image. If you just click SELECT on this control, the current image will be printed. But it has a menu mark; if you pull it down, you see a menu of `Print Snap` or `Options...` The last item has a window mark ('...'), so if you click on it an options sheet appears, like the one shown in Figure 9-4. This lets you adjust various options such as portrait or landscape mode.

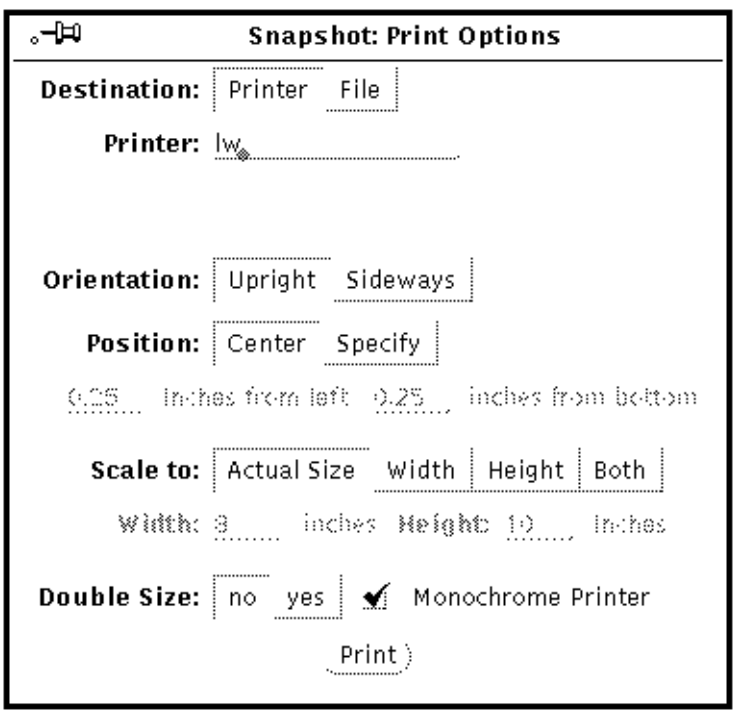


Figure 9-4. Snapshot: Print Options screen

There are numerous print options. You can have the image "printed" to a file, but the default is to print it to a real printer, whose name defaults to `lw` (LaserWriter, a generic PostScript printer). If you click on `File`, the `Printer` field is replaced by Directory and File fields. You can request that the snap be printed in portrait ("Upright") or landscape ("Sideways"), and either Centered or at a particular position. You can have the image printed actual size, double size, or you can specify Width, Height, or Both. Since at present most of us can afford only monochrome printers, that is the default. But if you have a color printer, feel free to check the Monochrome box off to print in color. In all cases, the program *rash*, a standard part of OpenWindows, is used to convert the image from Raster format into PostScript.

A quicker way of printing is to drag the image from the *snapshot* program's drag-and-drop target (the rectangle in the upper right of the main window) onto a copy of the *printtool* program (described in Chapter 7, *The OpenWindows DeskSet Clients*). This way you need only set your print options once, in *printtool*, and all your print requests that go through *printtool* will use these settings.

Snapshot works very nicely with normal X11 windows on OpenWindows. But Sun's OpenWindows also allows you to run NeWS and SunView clients. Because *snapshot* is an X11 program, it cannot know about either NeWS or SunView clients. NeWS client windows can easily be recorded by using *snapshot*'s *Snap Region* facility. However, SunView clients are totally invisible. Neither *xwd* (with the -root option) nor *snapshot* can "see" such clients. This is because of the way the sharing of the screen is implemented; see Appendix L, *Running SunView Applications on OpenWindows*, for details. What matters here is how we can get an image of such programs. The best way we've found so far is to use *screendump*, an ancient and venerable SunOS program that simply copies the physical contents of the frame buffer into a file on disk. Since *screendump* is not part of any windowing system, it doesn't care about NeWS, X11, SunView, or any difficulties or disagreements among them. It simply copies what is visible on your screen into a disk file for you. Let's say you want to get a screen image of the SunView *fontedit* program - this is how Figure 9-22 of the *fontedit* program at the end of this chapter was produced. First, start the program that you want to capture, and set its screen up the way you want it. Then, move the program's window† to the upper left of the screen. This is position (0,0), and makes the following steps easier. From a shell window, run

```
screendump > screen.rs
```

But that dumps the entire screen. If you only want one part of the screen, you can specify options that request part of the screen, or use a copy of *snapshot* with the *Grab Region* control to select the part you want. If you prefer command-line-based methods, you could use the PBMplus package, described later in this chapter, to cut out the part you need. For this example, we used

```
screendump -X 680 -Y 710 > fontedit.rs
```

† The SunView mouse conventions are different from those of OPEN LOOK; see Appendix L, *Running SunView Applications on OpenWindows*. Briefly: click and hold the middle pointer button on the titlebar to move a SunView window.

(note upper case X and Y) to just get the 680x710 region from the upper-left corner. Unfortunately, the 680 and 710 had to be derived experimentally, since *xwininfo* won't report on a SunView window, even in OpenWindows Version 3.0. For more information on *screendump*, see the reference manual page in your SunOS manual set.

9.1.2 xwd, xwud – Dump an X Window

If you don't have the OpenWindows DeskSet tools, you may wish to use the MIT standard client *xwd* to capture images in a "dump" format. *xwd* stores window images in a formatted window dump file. This file can be read by certain other X utilities for redisplay, printing, editing, formatting, image processing, etc.

To create a window dump file, type:

```
% xwd > file
```

The pointer will change to a small cross-hair symbol. Move the cross-hair pointer to the desired window and click any pointer button. The keyboard bell rings once when the dump starts and twice in rapid succession when the dump is finished.

To make a dump of the entire root window (and all windows on it), use the `-root` option:

```
% xwd -root > file
```

When you select a single window, *xwd* takes an image of the window proper. To include the titlebar, simply to click the button in the *olwm* titlebar instead of in the window.

xwd allows you to capture a single window or the entire root window. But what if you want an image that includes more than one window or parts of multiple windows? One method is to use *xmag* (described below) to capture an image of multiple windows and then use *xwd* on the *xmag* window! Since *xmag* is intended to magnify, if you want the window image to be the actual size, you must specify that no magnification is performed. To do this, you run *xmag* with the option `-mag 1`. See the *xmag* reference page in Part Three of this guide for more information. For OpenWindows users, a more flexible method of capturing part of the screen is to use the "Snap Region" option of *snapshot*, which we'll describe shortly.

To redisplay a file created with *xwd* in a window on the screen, use the *xwud* client, an undumping utility. Specify the dump file to display as an argument to the `-in` option:

```
% xwud -in file
```

When you get tired of looking at it, you can remove the image by pulling down and releasing *Quit* in the *Window Menu*. If you prefer, you can type Control-C (your INTR character) in the shell window from which you started *xwud*.

9.1.3 xpr, xdpr – Print an X Window

xpr takes as input an X Window System dump file produced by *xwd* and converts it to a printer-specific format that can be printed on the DEC LN03 or LA100 printer, a PostScript printer such as the Apple LaserWriter, the IBM PP3812 page printer, and as of Release 4, the HP LaserJet (or other PCL printers) or the HP PaintJet. By default, output is formatted for the obscure DEC LN03 printer. Use the `-device` option to format for another printer. For example, to format a window dump file for a PostScript printer, type:

```
% xpr -device ps file > file.ps
```

Other options allow you to change the size, add headers or footers, and so on. See the *xpr* reference page in Part Three of this guide for details.

You can use *xwd* and *xpr* together, using the standard UNIX pipe mechanism. For example:

```
% xwd | xpr -device ps | lpr
```

Because this is such a common operation, there is an *xdpr* shell script (csh) that rolls these three commands into one. *xdpr* accepts most of the options accepted by *xwd*, *xpr*, and *lpr* (1). Thus, you could use the command:

```
% xdpr -device ps
```

to take a window dump (`xwd`), convert that file to PostScript (`xpr -device ps`), and print the output (`lpr`). See the *xdpr* reference page in Part Three of this guide for more information.

If you routinely use *xdpr* with some device other than the now-defunct LN03, you can provide another default device by editing a copy of *xdpr*†

Hopefully a future version of *xpr* will allow use of X resources (see Chapter 12, *Setting Resources*) to specify the default printer.

Note that when you start piping together the output of X clients, you run into some ambiguities. For example, if you pipe the output of *xwd* to *xpr* and for some reason the *xpr* command fails, *xwd* will still be there waiting for pointer input. The original UNIX pipe mechanism doesn't have the concept of data dependent on pointer input! The integration of the UNIX model of computing (in which standard input and output are always recognized) and the window model is not always complete, sometimes leading to unexpected behavior.

As an even more flagrant example, you can create a pipe between two programs, the first of which doesn't produce standard output and the second of which doesn't recognize standard input. The shell doesn't know any better and the programs themselves go on their merry way with pointer and windows.

However, it is nice to know that you can pipe together program output, even when some of those programs may not produce output until you intervene with the pointer.

Even without pipes, you should start thinking about how these programs could work together. For example, the pictures of fonts in Appendix C, *Standard Bitmaps - X11, OPEN LOOK and OpenWindows*, were created by these steps:

1. Display a font with *xfd*. (See Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, for instructions on how to use xfd.).
2. Resize the window to improve readability, using the window manager.

† You may wish to change the initialization of the variable *xprv* from set xprv=() to set xprv=(-device ps) near the start of the file.

3. Create a window dump file with the command `xwd > file`.
4. Create a PostScript file from the dump with the command:

```
xpr -device ps file > file.ps
```

Print the PostScript file on a PostScript printer with the standard print command *lp* or *lpr* command.

Even though the UNIX shell will accept a pipe between *xfd*, *xwd*, and *xpr*, what actually happens is that *xwd* starts up faster than *xfd*, and is ready to dump a window before the *xfd* window appears.

9.1.4 Xloadimage

The *contributed client Xloadimage* does not save images, but will load nearly any graphics image, and present it on your X screen. To use it to view a file, say *face.xbm*, simply type

```
xloadimage face.xbm
```

If the file is a picture in any of the known formats, you will get a message like

```
face.xbm is a 47x64 X11 bitmap file titled 'UofT'
```

and then a window with that image in it will appear:

Figure 9-5. xloadimage face.xbm

In some bitmap formats a "title" is stored with each image, while in other formats it is not. If your bitmap file has a title, it will be printed in the message (as it was in this example) and displayed in the window's titlebar (though it will be truncated if the bitmap is small). If there is no title, the last part of the pathname (the "filename" part) will be used.

Multiple bitmaps may be displayed just by giving more than one bitmap file name on the command line. To display multiple bitmaps sequentially, you just give the filenames on the command line in the order you want them to appear†. To move from one bitmap to another, type a letter "n" in the bitmap display window.

For example,

```
xloadimage face1.xbm face2.xbm face3.xbm
```

This would display the bitmap from *face1.xbm*; to get at the second bitmap, move the pointer into the bitmap and type "n" for "next". There are a few other single-letter commands listed in the reference manual; the most useful is probably "q" for "quit."

† An earlier version of this program would combine multiple bitmaps given on the command line into a single bitmap. If you have this version, add `-slideshow` to display multiple bitmaps.

If the bitmap is too large to fit comfortably on your screen, *xloadimage* will display only a part of it. However, the program does not use scrollbars to move around. Instead, you "grab" the bitmap by pushing pointer button number one, and "drag" the display around. This does not conform to any standard (OPEN LOOK, Motif, or Athena), but in practice it works fine.

The number of formats supported by *xloadimage* is impressive, and it grows regularly because the program source code is structured to make it easy for C programmers to add code for new file types. Here is a recent list of the more common formats:

Table 9-1. Common XLoadImage Bitmap Formats

Formats Supported
FBM (Fuzzy BitMap library) Image
Sun Rasterfile
Sun Icon file
CMU WM Raster
Portable Bit Map (PBM, PGM, PPM)
USENIX/UUNET Faces Project
GIF Image
X Window Dump (from *xwd*)
X Bitmap
X Pixmap
Group 3 FAX Image
MacPaint Image

Some programmers have added their own proprietary formats as well; these are not in widespread use so they aren't shown here. It's fairly easy for a C programmer to add new types.

One problem with *xloadimage* is that it has difficulty with certain full-color images; it often fails to get the colors right. This will probably be corrected in a future release, though.

9.1.5 Xv

While *xloadimage* has a real breadth of file types, another contributed program called *xv* has a wide variety of operations. *Xv* as distributed can only read the GIF, PBM/PGM/PPM, XBM, SunRaster, JPeg and TIFF formats, as well as one local format called PM. But it can do quite a bit with images. It can re-scale an image. It can rotate images in 90-degree steps. It can "crop" images, throwing away extraneous sections of background. It can change the

coloring (use fewer colors, perform "gamma correction", etc.). It can write out images in the same formats it reads, plus a few others such as PostScript. And it can grab regions of the screens (but not particular windows). As the program's own author says:

"It slices, it dices, and it'll balance your checkbook if you aren't careful."

Let's run *xv* on the sample bitmaps we used with *xloadimage*. First, the face file. We'll give the command

```
xv face.xbm
```

and see how it looks:

Figure 9-6. xv face.xbm.

But this is rather hard to see, so let's resize it, using *olwm* to grab and drag one of the resize corners. When we let go, *xv* gets notified that its window is resized, so it scales the bitmap and redisplays it. For a large full-color bitmap this might take some time, but for this simple example it is almost instantaneous. The result is not impressive, because the underlying bitmap needs to be cleaned up. (We'll look at tools for doing so in the next section.)

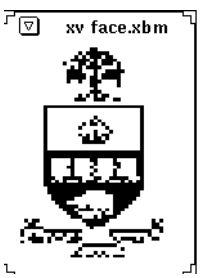


Figure 9-7. Bitmap after re-scaling.

When you want to do more to your image, you can click the right button (which is normally the MENU button in OPEN LOOK), and get the control screen. This program is not written to use OPEN LOOK; indeed, its control interface is more like that of the Macintosh™:

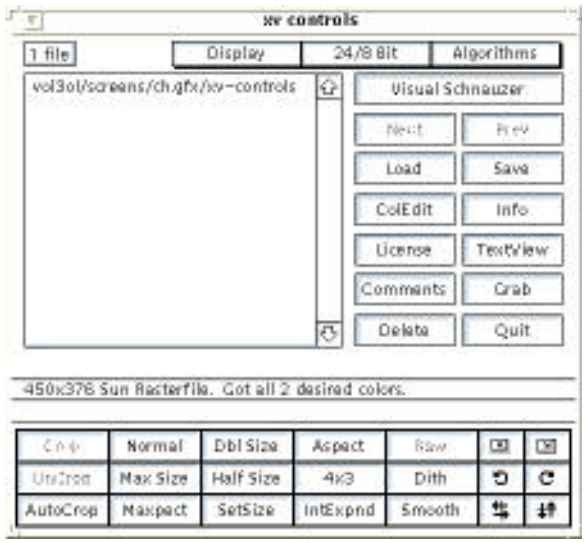


Figure 9-8. Xv controls

If you have multiple files, they will appear in the filename list. To the right of that list are buttons for *Next* and *Previous* (which are grayed out here because there is only one file), *Info* which pops up the Information window in Figure 9-9, *Save* which lets you save the file in any of several formats, and *Quit* which gets you out of *xv*.

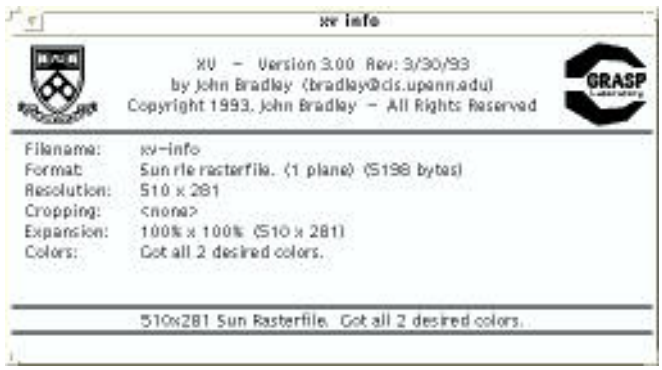


Figure 9-9. Xv Info Screen.

Notice the window mark; this is not a popup window, but another main window. If you want to get rid of this window, you can just click in it. Or, click on the *Info* button again. If

you select the *Save* button, the program will pop up another dialog box, the *xv save dialog*, shown in Figure 9-10.

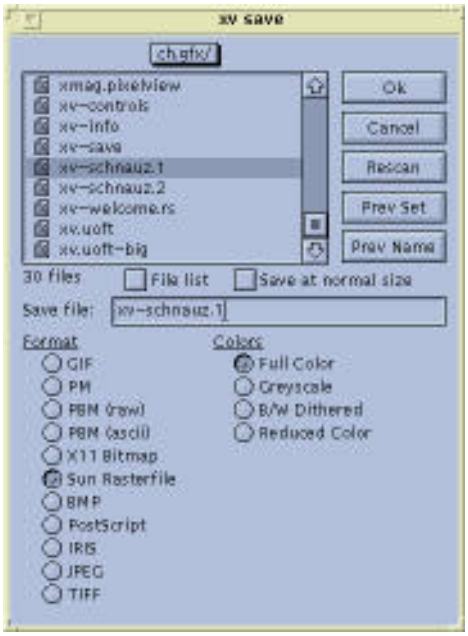


Figure 9-10. Xv Save Dialog.

Like the control panel, the save dialog is a full window, not a popup window, and it is more Mac-like than OPEN LOOK-ish. To select a format, for example, you click the left button (normally SELECT) in the circle beside the name of the format you want to use, and that circle gets a black circle inside it to indicate it is highlighted. This type of choice item is commonly called a “radio button”. An OPEN LOOK program would use either an exclusive choice or (more likely) an abbreviated choice, since you are unlikely to change formats very often.

To save a file, type its name into the *File name* text box. As soon as you start typing, the *Save* button becomes active, and can be clicked on when you have finished typing the file-name and chosen the size, format, and color model you want from the button selection panel.

The *Visual Schnauzer* gives a directory view similar to that of the Sun File Manager described in Chapter 4, *Using the OPEN LOOK File Manager*. Initially it appears as shown in Figure 9-11

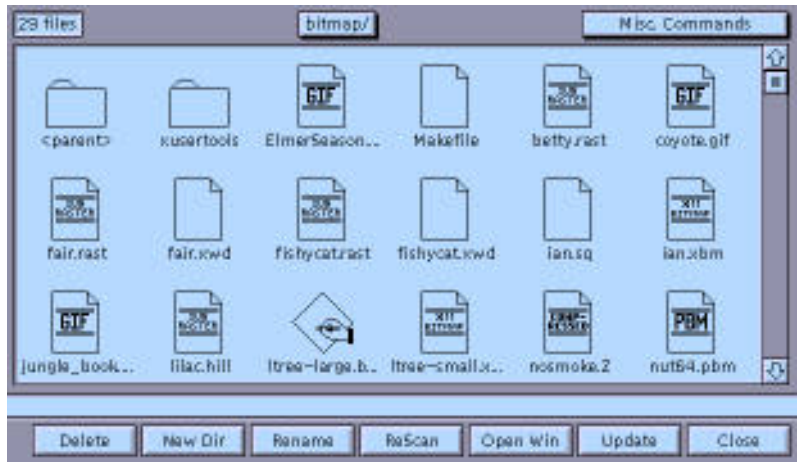


Figure 9-11. xv Visual Schnauzer

This in itself is fairly impressive. But it really comes to life when you notice that the box at the top right, labelled *Misc Commands*, is a pull-down menu (use the *Left* pointer button to activate it). If you select *Generate Icons* from this menu, any files in that directory that are

graphic files that xv can handle get replaced by iconic views of the pictures. The display now looks like Figure 9-12

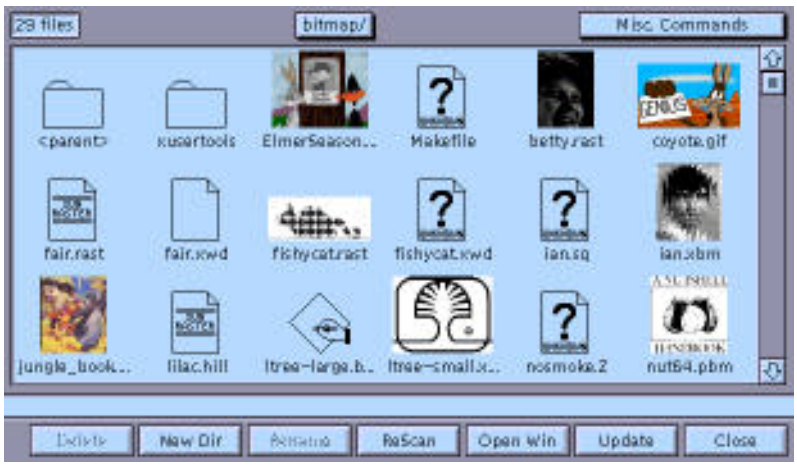


Figure 9-12. xv Visual Schnauzer with icons

This is a very powerful feature for browsing a directory of images. However, be aware tht the images are stored on disk (in a subdirectory called #), and can take up to 4Kb per image for color. When you're finished with them, you can remove the *.xvpics* directory, since it only takes a minute or two to regenerate.

9.2 Bitmap Editing and Conversion

This section looks at some programs that can be used for creating or modifying simple bitmap images such as those used by The X Window System for program icons, and by the OPEN LOOK GUI for *filemgr* icons.We also look at one more advanced program, *touchup*, for editing larger bitmap pictures. Finally, we take a look at a toolkit for converting bitmap images among dozens of different formats and otherwise massaging them.

9.2.1 iconedit (OpenWindows)

Iconedit is a bitmap editor that uses the OPEN LOOK GUI, so it is consistent with what you may expect, and it edits files in Sun "Icon" format in addition to XBM (X BitMap) and XPM (X PixMap) format. *Iconedit* is also similar in operation to *bitmap* and *olpixmap*, which will be described below. When started, *iconedit* puts up an empty edit window with numerous controls (see Figure 9-13). At this point, you can just move the pointer into the large drawing region and use the pointer buttons. The left (SELECT) button will turn individual points (pixels) on, while the middle (ADJUST) button will turn them off. Dragging

the pointer with either of these buttons depressed will continue the action, so you can draw simple figures just by holding SELECT and dragging the pointer.

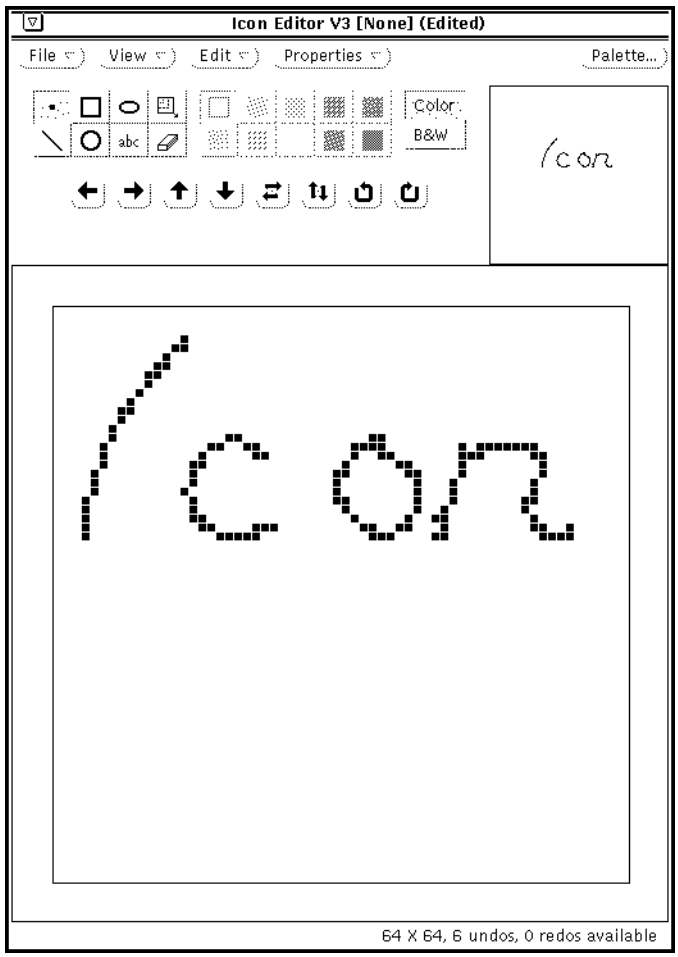


Figure 9-13. Iconedit start-up screen after a bit of freehand drawing.

The screen features Menu Buttons for File, View, Edit and Properties. File lets you load an existing image, save the screen image to a file, or print the screen image. The main window also has icons to select pixel drawing, lines, rectangles, circles, text ("abc"), region selection, and erasing. Unlike simpler programs such as *bitmap* (see Section 9.2.2, "bitmap" on Page 234), this program lets you add *text* in a variety of faces and sizes into your bitmap. These are controlled from the Text window, which appears when you click SELECT on the button with "abc" on it.

You can also select from one of several fill patterns by selecting one of the icons at the top right of the window. These will be used for filling rectangles, circles, or ellipses that you draw. The default is white, that is, no filling.

If you are on a color display, you can choose whether you want to edit a color or monochrome ("B&W") image. On a monochrome display, you can only edit monochrome images.

Iconedit also has the ability to all or part of move the bitmap up, down, left or right one pixel at a time (the four directional arrows), to flip it vertically or horizontally (the double arrows), and even to rotate the bitmap either clockwise or counterclockwise (the circular arrows). These can be very useful features.

For example, here is a simple demo icon, with the text window shown, just after we used the *Text* mode to put the words "My Demo" at the bottom:

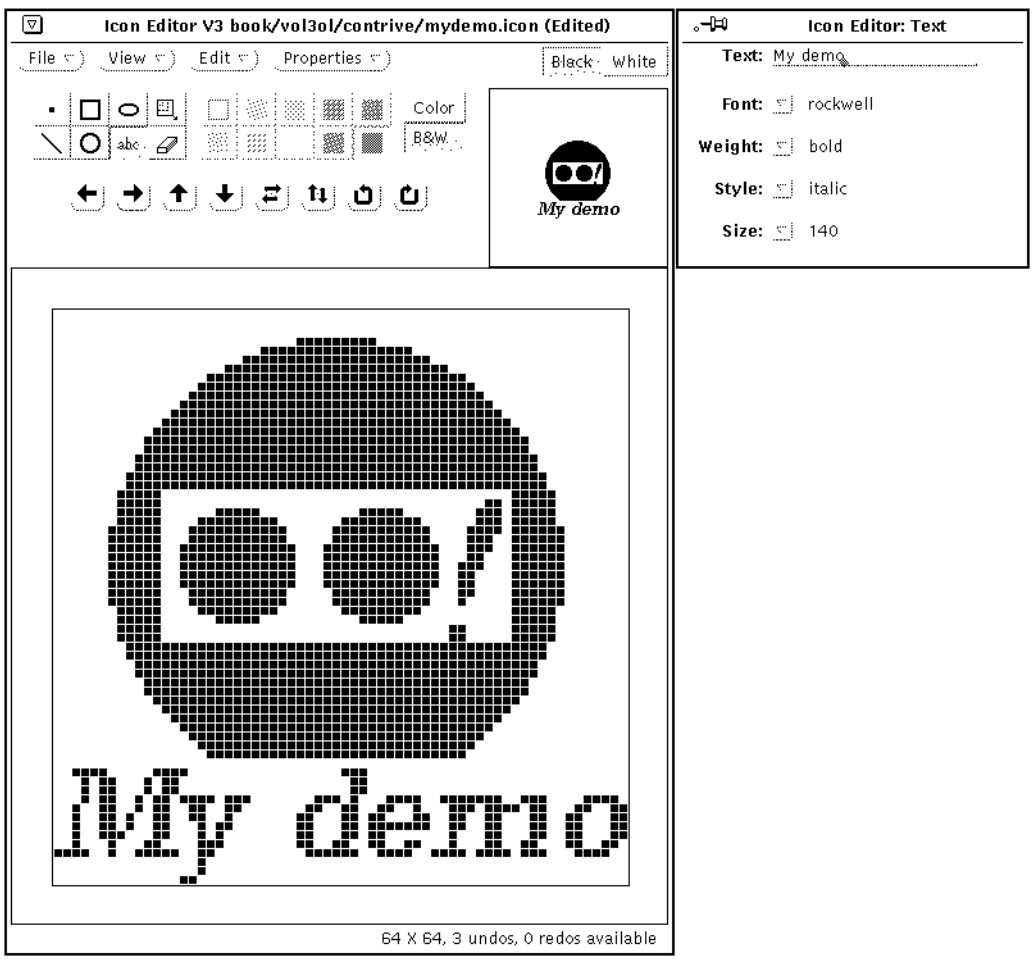


Figure 9-14. Iconedit in action

The *View* menu has only one item, that is *Grid on or off.* The Grid is a background of vertical and horizontal lines to help you align objects on the screen.,

The *Edit* menu (which is also the default menu when you press the MENU pointer button in the drawing area) lets you cut or paste an area of the screen, undo changes, etc. Notice that the OpenWindows L-keys, such as L6 for Copy, L8 for Paste, L10 for Cut, and L4 for Undo, all work in this context and indeed in most of *iconedit*.

The *Properties* menu does not get a Properties sheet, but lets you choose the format and size of the image. The known formats are XView Icon, X Bitmap, Color X Pixmap, and Monochrome X Pixmap. The sizes are shown in Table 9-2.

Table 9-2. IconEdit File Formats

Size	Format
16x16	Standard X Cursor
32x32	OpenWindows File Manager Icon
48x48	(unused)
64x64	*olwm* program Icon
128x128	(unused)

There is much more to *iconedit* than we have shown here. Try it out! *Iconedit* is a good choice for the OpenWindows user to edit bitmaps.

9.2.2 bitmap

If you have no OPEN LOOK package installed, you can use the *bitmap* program. the simplest and most widely available of the bitmap editing programs discussed in this section. It does not conform to the OPEN LOOK GUI nor to the OSF/Motif conventions, but it *is* included in the standard X11 software, so it is available almost anywhere that The X Window System goes.

Like *iconedit*, *bitmap* allows you to create and edit small *bitmaps*. You can use *bitmap* to create backgrounds, icons, and pointers. *bitmap* is primarily a programming tool for application developers. However, several applications allow you to design your own icon or background pattern with *bitmap*, save it in a bitmap file, and specify that filename on the command line.†

For example, *xsetroot* (described in Chapter 14, *Customization Clients*) allows you to specify a bitmap that will be used as the background pattern for the root window or as the root window pointer.

To invoke *bitmap*, type:

```
% bitmap &
```

† There are many bitmaps included in the X distribution. These can generally be found in the directory *$OPENWINHOME/share/include* or */usr/include/X11/bitmaps*. Samples are shown in Appendix C, *Standard Bitmaps - X11, OPEN LOOK and OpenWindows*.

If you provide a filename that contains an existing bitmap, it will be loaded (X11R4 required you to do so), otherwise a blank *bitmap* window is displayed, as shown in Figure 9-15.

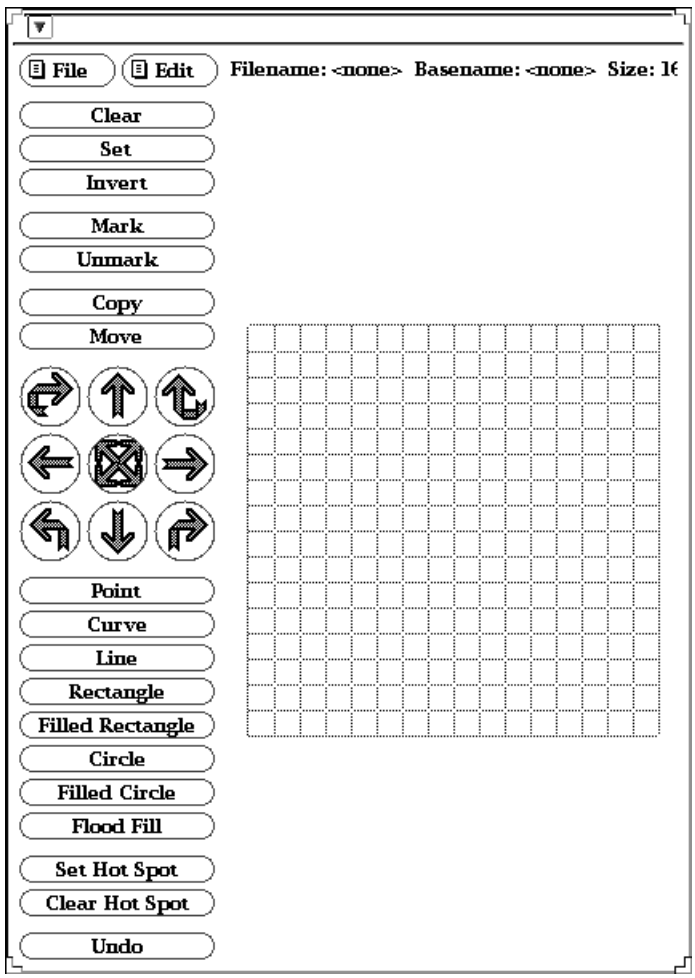


Figure 9-15. Bitmap window

The window that *bitmap* creates has three sections:

1. At the top is a partial “menu bar”, with a File and Edit menu and a display of the current filename (if any) and size (16x16, but it usually gets truncated).

2. The largest section is the checkerboard grid, which is a magnified version of the bitmap you are editing. The default size grid for new bitmaps is 16x16. (For existing bitmaps, the size of the grid will be the size of the bitmap. For example, if you open

a file containing a bitmap of size 100 pixels by 62 pixels, the grid will be 100 cells wide and 62 cells high.) If the cells in the grid aren't large enough for comfortable editing, resize the window. Each square on the grid will be enlarged proportionally.

3. On the left side of the window is a list of commands in command buttons that you can invoke with any pointer button.

If you want to create a new bitmap in a grid of different proportions than the default size 16x16 grid, you can specify ***WIDTH*x*HEIGHT*** on the command line after ***filename***. For example, to create a grid double the default size, enter:

```
% bitmap filename 32x32 &
```

The ***WIDTH*x*HEIGHT*** argument is used only when creating a new bitmap. Existing bitmaps are always edited at their current size; this program *cannot* re-size bitmaps.

Keep in mind that there is an interaction between the size of the bitmap being edited and the size of the *bitmap* window. By default, each cell in the *bitmap* editing area is 13 pixels square. If the bitmap being edited is large, this may result in an application window larger than the screen. (Since *bitmap* does not provide a scrollbar, a large window may make it impossible to edit!) Specifying an explicit size for the overall application using the `-geometry` option (or resizing it with the window manager) will change the size of the editing window because *bitmap* will automatically adjust the size of each editing cell to fit.

However, this type of adjustment has limitations. The *bitmap* application defines a minimum size of 3 pixels for the cells in the bitmap editing area. This means that in the smallest *bitmap* window you can create, each pixel in the bitmap itself will be represented by a cell 3 pixels square. Even if this adjustment creates a window that fits on your screen, it is extremely difficult to edit individual pixels represented by cells 3 pixels square.

Since we have covered the OPEN LOOK bitmap editors, we do not discuss the MIT *bitmap* editor in detail here. Consult the client reference page in Part Three of this manual for more details on the *bitmap* client.

9.2.3 touchup (SunView only)

Touchup is a sophisticated "draw" program, rivalling some of the Macintosh draw programs. *Touchup* is a contributed software application that at present is only configured for the SunView system.†

However, it will therefore run on early versions of OpenWindows, and is interesting for several reasons. It can edit much larger bitmaps than can the three text-mode bitmap applications described above. It has a wide range of brush types built in, and allows you to define your own, including by picking part of an existing bitmap for use as a brush. And it lets you specify write the bitmap out with a different size than you read in, which is useful for cropping or fitting bitmaps. (This is not a scaling operation, but a cutting or padding operation.)

† SunView is Sun's popular pre-X11 window system. This program may have been ported to Sun's XView toolkit by the time you read this, which would make it available on any X platform. However, this is not guaranteed to happen.

Since it's not (yet) a true application of The X Window System, we don't have space to describe the program in detail in this book. Here is a sample screen, editing my world-famous "fishycat" bitmap (Copyright © by Ian F. Darwin):

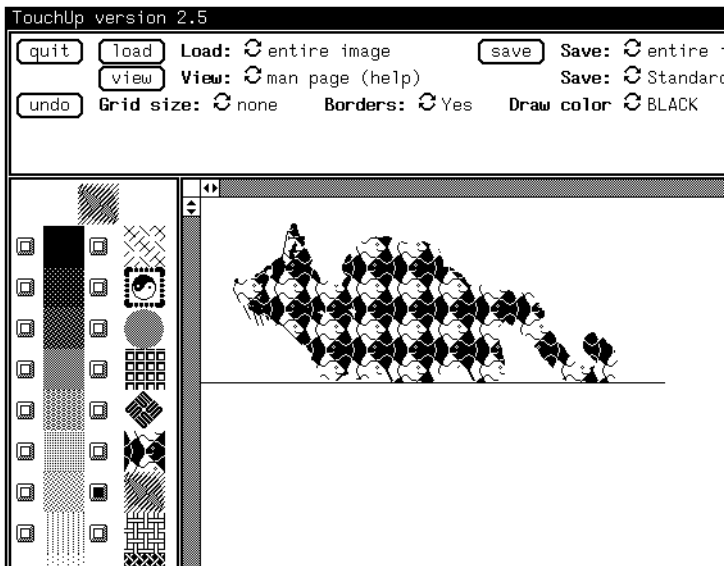


Figure 9-16. Touchup in action

Use of *touchup* will be obvious to anyone who's used a paint program; to anyone who hasn't, and wants to do bitmap editing, I recommend getting a copy of this program (see the Preface) and learning it.

9.2.4 Magnifying Portions of the Screen: xmag

Although it doesn't actually capture part of the screen for you, *xmag* is a useful little program for displaying part of the screen. The *xmag* client enables you to magnify a portion of the screen. The close-up look *xmag* affords can assist you in creating and editing bitmaps and other graphic images.

xmag is primarily a tool for application developers using sophisticated graphics programs. But you could also use *xmag* in concert with the *bitmap* client. For instance, say you're running a program that creates a special image on the root window and you'd like to create a *bitmap* file of a part of that image. You can display a magnification of the image you want with *xmag* and try to recreate the image by editing in an open *bitmap* window.

If you invoke *xmag* without options, you can interactively choose the area to be magnified (the *source* area) and position the magnified image on your screen. At the command line, type:

```
% xmag &
```

The pointer changes to a small cross (the cross-hair cursor) in the center of a small, hollow square with a wavering border. (By default, the square is 64 pixels on each side.) Move the cross-hair cursor, placing the square over the area you want to magnify, and click the first pointer button.

The hollow square becomes enlarged to the size of the magnified image. (By default, the image is magnified five times.) By default, *olwm* places the *xmag* window containing the magnified image in the upper-left quadrant of the display. Let's say we want to see the icon of *snapshot* blown up. Conveniently, *xmag*'s default viewing size is 64x64, the same as OpenWindows icons. Center the active area over the icon, and click any pointer button, and you should see a window similar to that shown in Figure 9-17

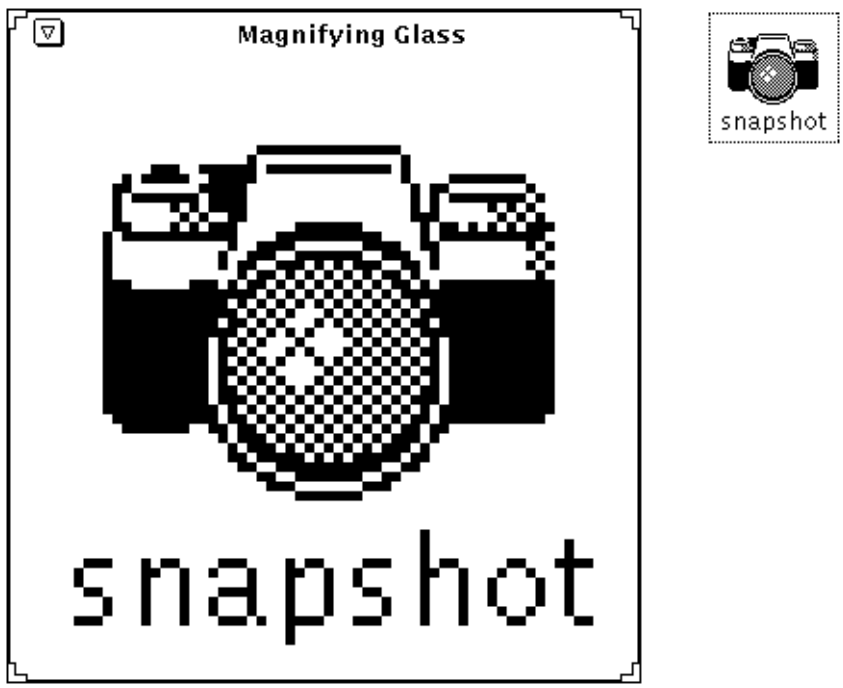


Figure 9-17. xmag window displaying magnified screen area.

The string "Magnifying Glass" will be displayed in the *xmag* window titlebar. This is the default title string of the application, and unlike most programs, *xmag* provides no way to change it.

The default size *xmag* window shows an area 64 pixels square, magnified five times. This magnification enables you to see the individual pixels, which are represented by squares of the same color as the corresponding pixels in the source image.

Rather than use the default source area and magnification, you can specify other values on the command line. See the *xmag* reference page in Part Three of this guide for a complete list of options. Quitting xmag

As a shortcut to quitting the *xmag* program, you can type q, Q, or Control-C in the *xmag* window.

9.2.4.1 What xmag Shows You

xmag enables you to determine the x and y coordinates, bitmap bit setting, and RGB color value of every pixel in the *xmag* window. (See Chapter 11, *Command-line Options*, for a discussion of the RGB color model.) If you move the pointer into the *xmag* window, the cursor becomes an arrow. Point the arrow at one of the magnified pixels and press and hold down the first pointer button. A banner across the top or bottom edge of the window displays information about the pixel, as shown in Figure 9-18.

I

Figure 9-18. Displaying pixel statistics with pointer in xmag window

The banner displays the following information about the specified pixel:

- The x and y coordinates relative to the window. The default *xmag* window is, in effect, a grid of 64 squares on each side. Therefore, each pixel has x,y coordinates between 0,0 and 63,63.

- The bitmap bit setting. This is either 0 if the pixel is in the background color or 1 if the pixel is in the foreground color.
- The RGB value. This is a 16-bit value. The RGB specification is in three parts (of four hexadecimal digits each), corresponding to the three primaries in the RGB color model.

If you are trying to create a graphic image on a grid (such as the *bitmap* client provides), the x and y coordinates of each pixel can be especially useful. Also, the 16-bit RGB value specifies the color of each pixel with moderate precision. Depending on the number of colors available on your display, you can learn to use RGB values to specify an enormous range of colors (other useful tools for learning RGB values are the colormap editor in *xv* (described above) and also *xcoloredit).*

xmag provides these pixel statistics dynamically. If you continue to hold down the first pointer button and drag the pointer across the window, the banner will display values for each pixel as the pointer indicates it.

9.2.4.2 Dynamically Choosing a Different Source Area

If you want to magnify another portion of the screen using the same source area size and magnification, you do not have to start *xmag* again. Simply move the pointer into the *xmag* window and click the ADJUST pointer button, or press the space bar. The magnified image disappears and the cursor again becomes a cross-hair surrounded by a hollow square. Move the cross-hair cursor, placing the square over the new source area you want to magnify, and click any pointer button. The magnified image is immediately displayed in the same location as the first image.

9.2.5 The Portable Bitmap Toolkit

Unlike the on-screen editing provided by the above bitmap editing programs, the Portable Bitmap Toolkit (in the user-contributed distribution) provides dozens of non-interactive utilities for converting graphics files to and from portable formats. Developed by Jef Poskanzer, the Toolkit is composed of four parts, three of which correspond to a particular portable format:

- PBM: utilities to convert files to and from portable bitmap format.
- PGM: utilities to convert files to and from portable graymap format (grayscale images).
- PPM: utilities to convert files to and from portable pixmap format (color images).

The fourth part of the toolkit, PNM, provides utilities to manipulate images in any of the three formats. For example, the program *pnmenlarge* enlarges a portable "anymap" by a factor you supply. *pnminvert* inverts an image in any of the three portable formats.

The available utilities and the conversions they perform are summarized in the *README* file in the source directory. Some representative conversion utilities and their functions are listed in Table 9-3.

Table 9-3. Some PBM Toolkit Conversion Utilities

Utility	Converts
giftoppm	GIF to portable pixmap.
ppmtogif	Portable pixmap to GIF.
ppmtoxwd	Portable pixmap to X11 window dump.
xwdtoppm	X10 or X11 window dump to portable pixmap.
ppmtopgm	Portable pixmap to portable graymap.
fstopgm	Usenix FaceSaver file to portable graymap.
pgmtops	Portable graymap to Encapsulated PostScript.
pgmtopbm	Portable graymap to portable bitmap.
pbmtomacp	Portable bitmap to MacPaint.
macptopbm	MacPaint to portable bitmap.
pbmtoxbm	Portable bitmap to X11 bitmap.
pbmtox10bm	Portable bitmap to X10 bitmap.
xbmtopbm	X10 or X11 bitmap to portable bitmap.
pbmtoxwd	portable bitmap to X11 window dump.
xwdtopbm	X10 or X11 window dump to portable bitmap.

As the table indicates, some of the available utilities come in pairs—they can be used to convert a file to a portable format and back to its original format again. (The table also includes a group of three related utilities to convert X10 and X11 bitmaps to portable bitmaps and back again.)

Certain conversions can only be performed in one direction. For example, you can convert a portable graymap to a portable bitmap (using *pgmtopbm*), but you can't convert a bitmap to a graymap. The one-way conversions generally involve changing a file to a simpler format.

You'll probably be most interested in converting graphics files to formats suitable for use with X, namely X11 bitmaps or window dump files. Keep in mind that a portable bitmap has a different format than an X11 bitmap. The program *pbmtoxbm* converts a portable bitmap to a bitmap compatible with X11.

The conversions you may want to perform can be simple (directly from one format to another) or complex (through several intermediate formats). An example of a simple conversion is changing a portable pixmap to a portable graymap using *ppmtopgm*:

```
% ppmtopgm pixmap > graymap
```

The PBM Toolkit source directory includes a file called *TIPS* that provides helpful hints on using the utilities. Based on these suggestions, we performed a fairly complex conversion: a Usenix FaceSaver image to a bitmap suitable for use with X. The following command performed the conversion on the file *myface* to create *myface.bitmap*:

```
fstopgm myface | pnmenlarge 3 | ppmscale -yscale 1.125 | ppmtopgm |\
            pgmnorm | pgmtopbm | pbmtoxbm > myface.bitmap
```

Notice that this particular conversion requires seven utilities! This procedure is by no means intuitive. We relied heavily on the *TIPS* provided.

The seven conversions performed are:

- Convert FaceSaver image to portable graymap (`fstopgm`).
- Enlarge a portable anymap three times (`pnmenlarge 3`).
- Scale pixels in y dimension; x dimension is adjusted accordingly (`ppmscale -yscale 1.125`). This program produces portable pixmap output.
- Convert portable pixmap to portable graymap (`ppmtopgm`).
- Normalize contrast of portable graymap (`pgmnorm`).
- Convert portable graymap to portable bitmap (`pgmtopbm`).
- Convert portable bitmap to X11 bitmap (`pbmtoxbm`).

Be aware that the command:

```
ppmscale -yscale 1.125
```

may not be necessary on all systems or the necessary arguments may vary. If you omit *ppmscale* and the command is necessary, the system should return a message to that effect and also tell you what arguments to use.

The possible uses of the PBM Toolkit programs and the ways in which they can be combined are extremely varied. You'll have to do some experimenting. To orient yourself, read the files *README*, *TIPS*, and *FORMATS* in the source directory. The source directory also includes reference pages for each utility.

9.3 Commercial Desktop Graphics Offerings

As The X Window System becomes more popular, the variety of commercial offerings available for it will continue to grow. This is especially true in the graphics area. Here are some current offerings.

9.3.1 Arts & Letters

A Sun OPEN LOOK version of this popular MS-Windows desktop graphics tool was released in 1991. It offers scaling, shading, freehand drawing with Bezier curves, vectorization ("auto tracing") of bitmaps, numerous special effects including fitting text to arbitrary shapes, and a library of some 5,000 clip-art images, with many more available. Contact the vendor: Computer Support Corporation, 15926 Midway Road, Dallas TX 75244, phone 214-661-8960.

9.3.2 IslandPaint, IslandDraw

Island Graphics distributes the Island Productivity tools, which includes IslandWrite, IslandDraw, and IslandPaint. The former is a publishing package, while the latter two are graphics packages that integrate with it. Contact the vendor: Island Graphics at 415-491-1000.

9.3.3 Artisan

Artisan is a graphics package for manipulating bitmap images, that is, a full-featured "Paint" program. Contact Media Logic at 213-453-7744.

9.3.4 Adobe Illustrator

A Motif version of Adobe Illustrator is available for Solaris 2.3 and later. Contact Adobe Systems, Inc. Their World Wide Web address is http://www.adobe.com.

9.3.5 Corel Draw

A Motif edition of this full-function desktop graphics package is available from Corel Systems.

9.4 PostScript Viewing and Editing

Many systems in use today support the PostScript™ language. It provides a largely device-independent imaging model that can be used by printers or displays to produce attractive text and graphics. It was initially made popular by the Apple Macintosh. Lately, with the rise of printers with reasonable resolution (600 dpi and up), PostScript has even begun to be used in some commercial graphics and typesetting.

Since a large number of Suns have "Sun LaserWriter" printers (a private label for the Apple LaserWriter), there is considerable support for PostScript in current versions of SunOS. In fact, many of the OpenWindows tools discussed in Chapter 7, *The OpenWindows DeskSet Clients* assume that they can print to a PostScript printer. And Sun provides a software product, NeWSprint† that provides PostScript support on printers that lack it, such as the common Hewlett-Packard LaserJet™ and even some dot-matrix printers. Because of the relative slowness of printers compared with graphics screens, however, it is common to want to "view" a PostScript file before committing it to paper. That is what

† NeWSprint runs only on SPARC systems running SunOS4.1 or later and OpenWindows.

PostScript Previewing is about. Let's look first at Sun's *pageview* previewer, and then at some other previewers.

9.4.1 Pageview (OpenWindows only)

Pageview is a PostScript viewer that depends on PostScript features of the server included with Sun's OpenWindows offering. As you'd expect, Openwindows versions up to 3.2 use the NeWS PostScript system, while 3.3 and later use Display PostScript. It will not work with other servers, such as the standard X11R6 server (users of such systems that need PostScript should see the section "Other PostScript Viewers" below). It should work with other servers that include the Display PostScript extention, however..

Pageview can be started from the *olwm* root menu, or from the command line. If it is invoked with no filename, it waits for you to provide a file. It displays this message:

```
Use File->Load... to load a PostScript file
 or
use File Manager to drag and drop one from $OPENWINHOME/demo/PostScript
```

As the message says, you can either use the *File* menu's *Load* item to load a specific file, or you can use OPEN LOOK's drag-and-drop mechanism to tell *pageview* what file to read. On the other hand, if *pageview* is invoked with a filename, it will read that file. As a special case for use in pipelines, it accepts the sporadically-enforced UNIX convention that a filename of a single minus sign ("-") means the standard input, so you can say

```
troff -ms myfile | troff_to_ps | pageview -
```

to run a *troff* and some troff-output-to-PostScript converter, and feed the result of that into *pageview.*†

As has been mentioned, the *File* menu has a *Load* option that lets you load one PostScript file at a time. It also has a *Print* option that lets you send either the current page or the whole document to a PostScript-capable printer.

Pageview's normal operation when it starts up is to display the first page of the document, and wait for you to do something. You can move around, if the page is bigger than the view window. However, the present version does not use scrollbars; you drag the canvas around by clicking and holding SELECT, then dragging the pointer in the direction you want to move the canvas. Or you can move to different pages by using the *View* menu, which lets you move to the First, Last, Next (default) or Previous page.

The *Edit* menu lets you edit either *PostScript* or *Properties.* The *Edit->Postscript* option lets you edit the PostScript input, assuming that you know the PostScript language well enough to make changes. The editing facility is identical with that used in *cmdtool* and *textedit*, which are described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*. *Edit->Postscript* is also a great learning tool if you are teaching yourself the PostScript language. You can make a few changes and re-run the file, and see right away what effect your change has. For example, start up *pageview* with no options, and you will see this:

† Several companies provide versions of *troff* that produce PostScript; one such offering that works with Pageview is from SoftQuad Inc., 56 Aberfoyle Crescent, Toronto, Ontario CANADA M8X 2W4, phone 416-239-4801, 800-387-2777 (from U.S.A. only). E-mail mail@sq.com.

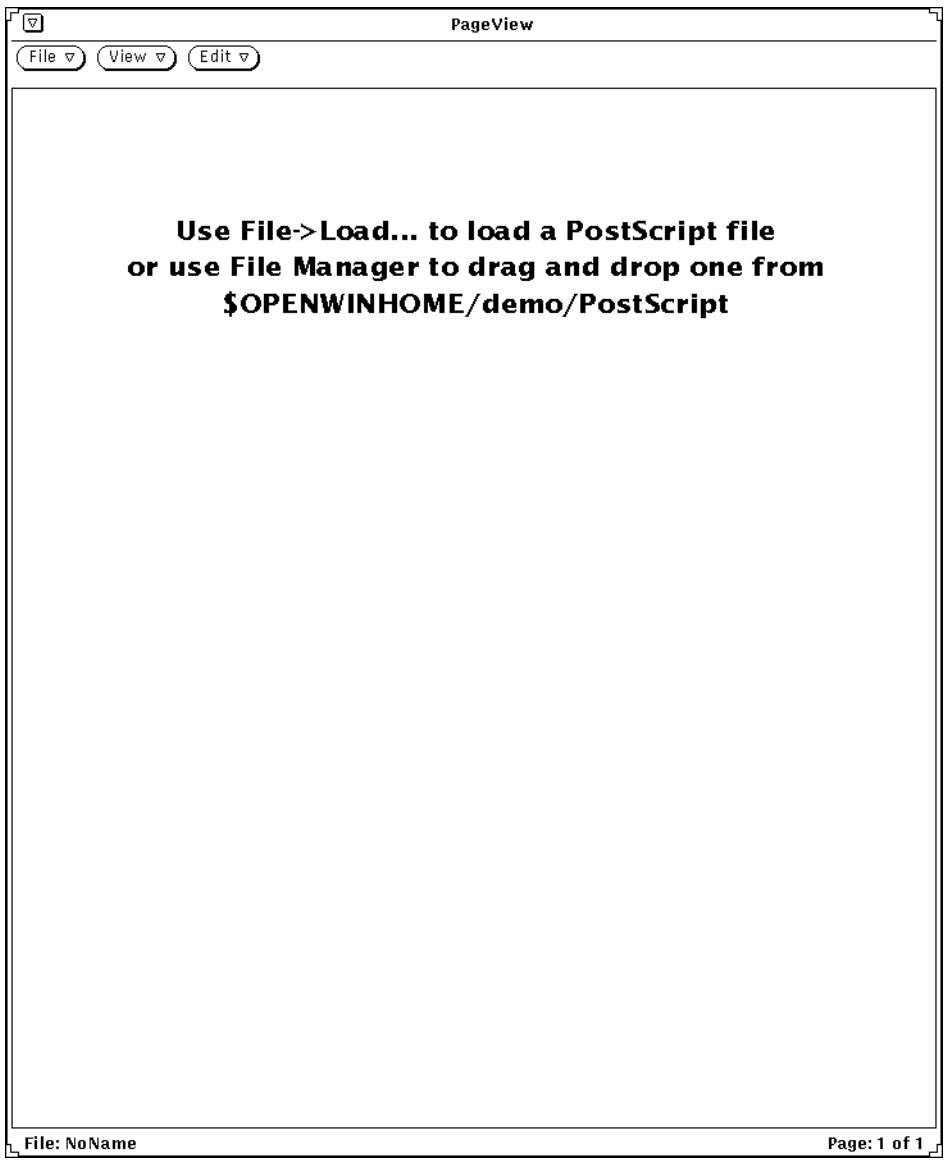


Figure 9-19. Pageview welcome screen

Just click SELECT on the *Edit* button, since *Edit->Postscript* is the default, and you will get an edit window:

Figure 9-20. Pageview edit window

Type a few lines like those shown,†and click on *Run*. The main window will display the output:

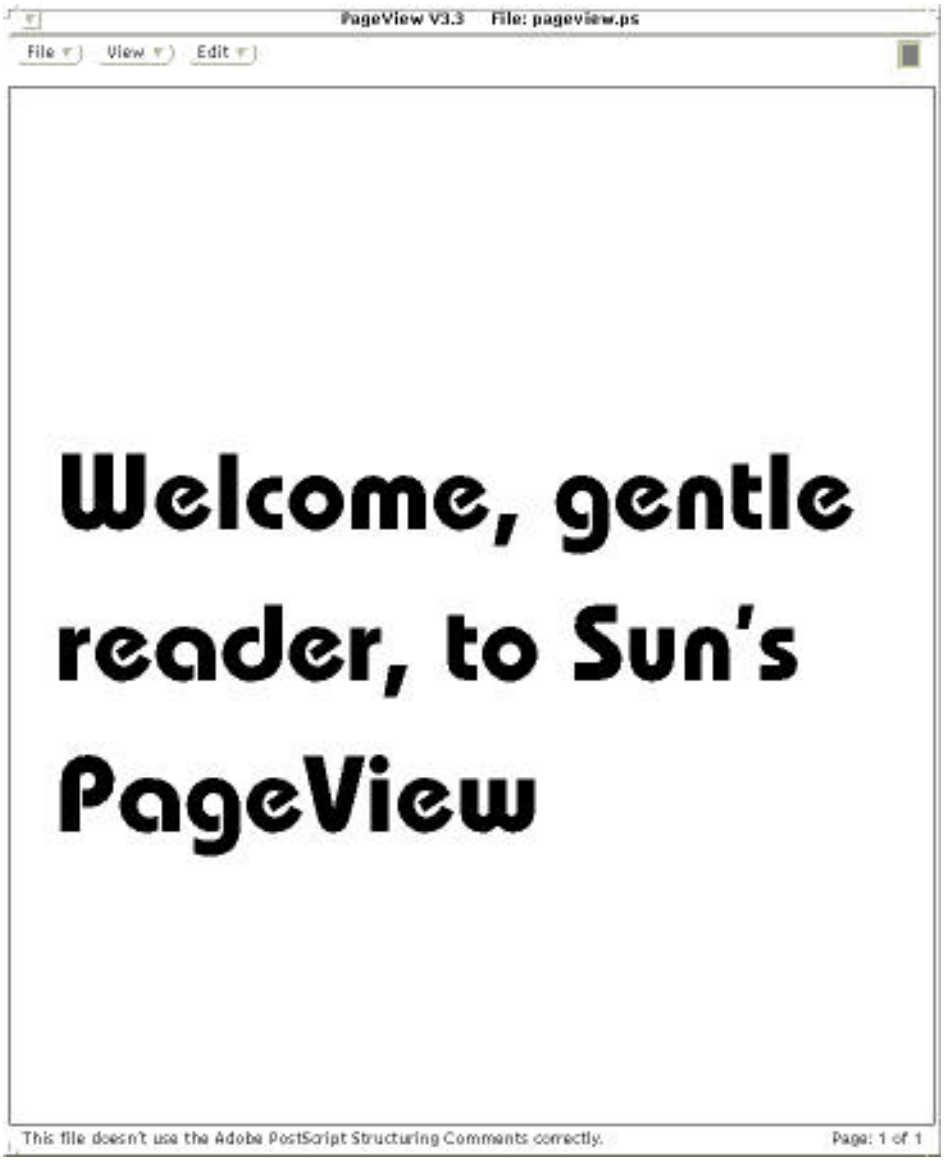


Figure 9-21. Output of pageview.

† Bauhaus is a commercial Type1 font; if you don't have it, use `Times-Roman` or some other font.

The *Edit->Properties* lets you change some settings, such as the size of paper, relative dots-per-inch, page orientation, etc. One useful mode is "anti-aliasing", which produces better quality type. These changes are not saved in a file; they only apply to the current *pageview* session; many of these can also be specified on the command line.

Pageview's command line options are detailed in the reference page in Part Three of this Guide. The most commonly-encountered difficulties with *pageview* in Release 2 of OpenWindows had to do with impure PostScript. There are several levels of "standardization" for PostScript. The base language has two versions (Version 1 is in use; Version 2 has recently been released by Adobe but isn't widely used yet). As well, "proper" PostScript documents include something called "structure comments", and in OpenWindows Release 2 and 3 the *pageview* program depended on them. In the current release of OpenWindows, *pageview* will work with or without the structure comments; if they are missing, it will read only the base PostScript language.

9.4.2 Other PostScript Viewers

Some system vendors include a PostScript interpreter called Display PostScript™ as part of their X servers. Display Postscript is in some ways patterned after the NeWS component of Sun's OpenWindows, but the two implementations of the PostScript language are, alas, not compatible. So you can't use the 2.2 or earlier *pageview* to display output on an IBM RS-6000 with AIX nor on a DECstation with Ultrix. However, these systems do provide their own PostScript viewers. Users of these systems should consult their vendor documentation.

In 1992, Sun Microsystems signed an agreement with Adobe Systems, the originators of the PostScript language and the vendors of Display Postscript, to make Display PostScript available in OpenWindows. Current releases of OpenWindows support Display PostScript rather than NeWS..

Finally, there are two programs in the "contributed software" category that can interpret PostScript and display it on an X11 window. These programs, *psview* and *ghostscrip/gsviewt*, are included in the CD-ROM accompanying this book, and available from many Internet archive sites including UUNET (see description of UUNET in the Preface). We do not provide any further information, as these programs are still undergoing change.

9.5 Font Editing

Since this chapter is about graphics, it would be incomplete without some mention of one of the most basic graphics that we use in our daily lives, that of text fonts. X11 does not include any official font editing facilities. The contributed program *xfed* provides minimal facilities for producing bitmap fonts in the X11 "bitmap distribution format" (BDF fonts). BDF is an X Consortium standard, and any implementation of X11 is supposed to either accept them (directly, as the DECwindows™ server does), by conversion using *bdftosnf* (as the MIT X11 servers do), or by conversion to its own format (as the older OpenWindows server does, using *convertfont*). Fonts in this format can therefore be used on any reasonable implementation of The X Window System. More detail on these and other font

conversion programs can be found in Volume Eight, *X Window System Administrator's Guide*.

The Sun operating systems (SunOS 4 only) includes *fontedit*, a bitmap editor originally written for SunView™, and unfortunately not yet (as of OpenWindows Version 3) converted to run on X11 for OpenWindows. It, too, provides simple editing of bitmap fonts. In usage, both *xfed* and *fontedit* are similar in flavor to *iconedit* or *bitmap*, but with extra controls for editing the font information. Figure 9-22 is a picture of the (SunView version of the) *fontedit* program in action

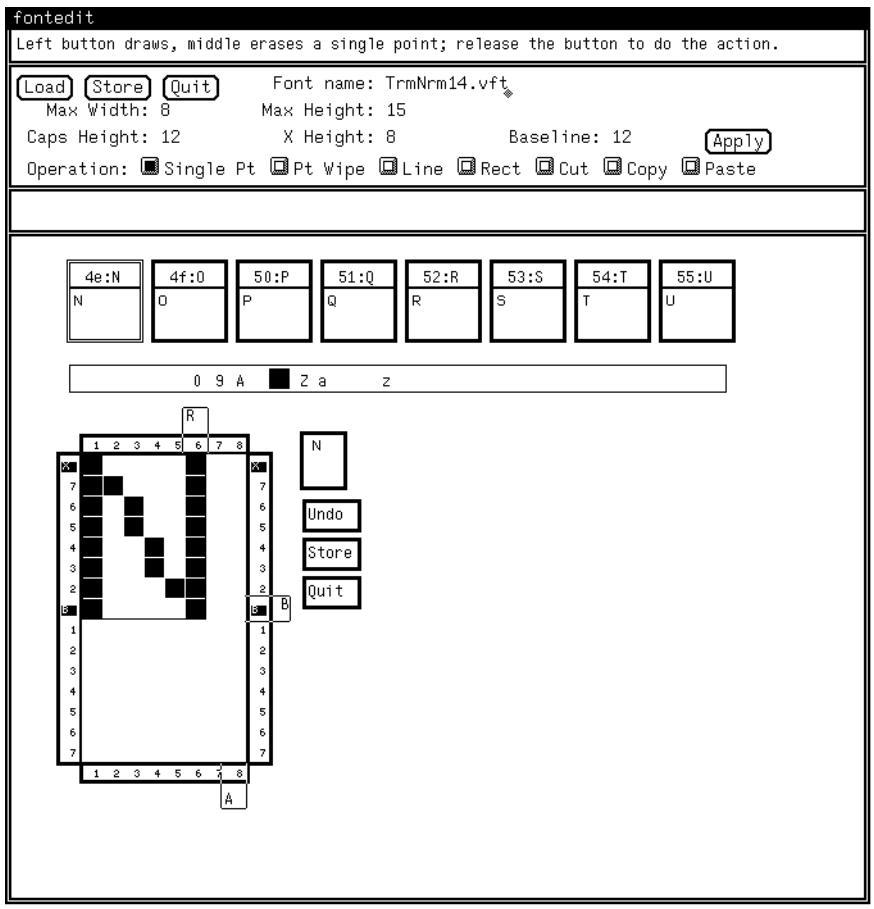


Figure 9-22. SunView fontedit.

For commercial production of fonts, there are several schemes in use. Most of the large commercial type foundries either have their own software, or use a package called *Ikarus* (from URW Software AG). *Ikarus* lets you edit outline fonts, and produce either scalable fonts or bitmaps in a variety of sizes and formats. As well, many companies are now producing software that helps you generate PostScript™ fonts. OpenWindows users who wish

to pursue this should investigate Sun's *Typemaker*™ offering, which makes fonts in an outline format ("F3") that the OpenWindows server can use. SunPics produces a CD-ROM called "Printer's Palette" that contains a variety of fonts and other tools for use with OpenWindows and its printing software package NeWSprint (described in Chapter 7, *The OpenWindows DeskSet Clients*).

Current versions of The X Window System (including OpenWindows versions that use Display PostScript) support the use of "Type 1" PostScript fonts. These can be created using tools that are popular on the Macintosh, such as *Fontographer* and *Font Studio*. These could be run under Apple's Macintosh Application Environment under Solaris 2, or you could use them on a Mac and convert the font files to "MS Windows" format and install them on your OpenWindows server, as discussed in Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*.

PART TWO: Customizing X and OpenWindows

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 10

Font Specification

Some OpenWindows client programs allow you to choose the fonts you want by selecting from a menu. Any user-ready commercial application, such as the DeskSet clients described in Chapter 7, *The OpenWindows DeskSet Clients*, and commercial publishing software packages such as *FrameMaker* and *Interleaf*, or graphics packages such as Adobe *Illustrator* or Corel *Draw*, provide a simple interface for selecting fonts. However, the *cmdtool* terminal emulators does not have such a menu, and the *xterm* terminal emulator only lets you choose from a few pre-specified fonts. You can choose the font used for the text in *olwm* menus or in *cmdtool* or *xterm* windows, but you must do it via the command line or from X resources..

The X Window System has a fairly complex font naming system which, like most things about X, is designed for maximum flexibility rather than for simplicity or ease of use. This wouldn't be so bad if a typical font name weren't mind-bending at first glance. To create a *cmdtool* window whose text is to be displayed in 14-point Courier bold, you'd hope to be able to type something like this:

```
% cmdtool -fn Courier-Bold-14
```

Fortunately you can do that on OpenWindows because of Sun's clever alias naming and "font scaling." Accordingly, users of OpenWindows may wish to skip this chapter at first blush, and return to it later when they need to select fonts via command line options or resource specifications, for MIT or other non-commercial applications.

On a normal X11 server, you might need to type, as a worst case, something like this:

```
% cmdtool -fn -adobe-courier-bold-r-normal--14-140-75-75-m-
        90-iso8859-1
```

Fortunately, you can use asterisks as wildcards to simplify this name to a somewhat more reasonable one:

```
% cmdtool -fn '*courier-bold-r*140*'
```

and you can define even simpler aliases, so that you could end up typing a command line like the simplified one shown earlier:

```
% cmdtool -fn Courier-Bold-14
```

In this chapter, we're going to try to make sense out of the sometimes bewildering jungle of information about fonts under X. First, we'll explain the font naming convention in detail. Along the way, we'll acquaint you with the appearance of some of the basic font families (groups of related fonts), and the various permutations (such as weight, slant, and point size) within each family.

Then, we'll talk about how to use font name wildcards to simplify font specification. We'll also talk about the font search path (the directories where the font files are stored), and how to define aliases for font names.

Finally, we'll talk about some of the utilities provided for dealing with fonts:

- *xlsfonts*, which lists the names of the fonts available on your server, as well as any aliases.
- *xfd* (font displayer), which allows you to display the character set for any individual font you specify on the command line.
- *xfontsel* (font selector), which allows you to preview fonts and select the name of the one you want (this name can then be pasted onto a command line, into a resource file, etc.)
- *text* (OpenWindows only), which allows you to display a representative text sample (character display) for any font selected from a menu of all available fonts.

10.1 Font Naming Conventions

The X11 "logical font naming convention" has been around since X11 Release Three, so it is included in all versions and variants of the X Window Systems in use today. As we'll see in a moment, these logical font names allow for complete specification of all of the characteristics of each font. Unfortunately, this completeness makes them somewhat difficult to work with, at least until you learn what all the parts of the names mean, and get a handle on which parts you need to remember and which you can safely ignore. (By the end of this chapter, you should have that knowledge.)

The *xlsfonts* client can be used to display the names of all the fonts available on your server. When you run *xlsfonts*, you'll get an intimidating list of names similar to the name in Ref f. Upon close examination, this rather verbose name contains a great deal of useful information: the font's developer, or foundry (Adobe); the font family (Courier); weight (bold); slant (oblique); set width (normal); size of the font in pixels (10); size of the font in tenths of a point (100 tenths of a point, thus 10 points); horizontal resolution (75 dpi); vertical resolution (75 dpi); spacing (m, for monospace); average width (60—measured in tenths of a pixel, thus 6 pixels); and character set (iso8859-1).

As mentioned earlier, font name wildcarding can eliminate lots of unnecessary detail. If you are already familiar with font characteristics, skip ahead to the section "Font Name

Wildcarding," later in this chapter, for some tips and tricks. If you need a refresher on fonts, read on as we illustrate and explain each of the elements that make up the font name.

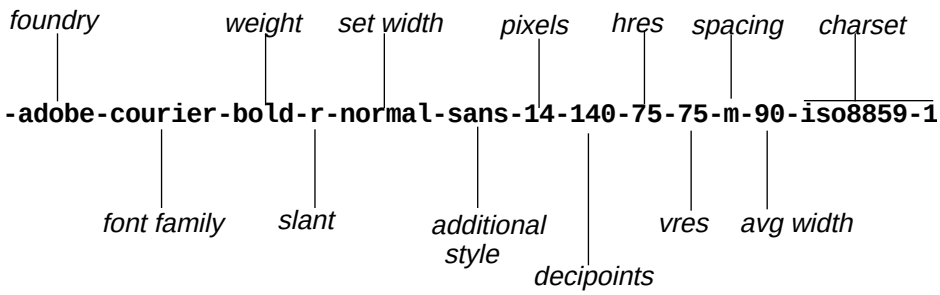


Figure 10-1. Font name, X11 Logical Font Naming Convention

10.1.1 Font Families

It has been a decade or so since the advent of desktop publishing and, by now, it is unlikely that anyone in the computer industry is unaware that text can be displayed on the screen and printed on the page using different fonts.

However, the term *font* is used somewhat ambiguously. Does it refer to a family of typefaces (such as Times Roman or Helvetica), which comes in different sizes, weights, and orientations? Or should each distinct set of character glyphs be considered a separate font?

OpenWindows takes the former approach, while historically X11 has taken the latter. When the documentation says that X11 Release 4 has more than 400 fonts , this sounds either intimidating or impressive, depending on your mood. But, in fact, the X11R4 distribution includes only eight font families (Courier, Helvetica, New Century Schoolbook®, Symbol, Times, Lucida® and the Clean family of fixed-width fonts), plus several miscellaneous and special purpose fonts. By contrast, both the Macintosh and OpenWindows support dozens of font families. Although OpenWindows comes with "57 fonts" (see Figure 10-2), there are actually about two dozen families when you remove variations such as Bold and Italic (AvantGarde (two versions), Bembo, Bookman, Courier, Gill, Gill Sans, Helvetica, Lucida, Lucida Bright, Lucida Sans, Lucida Sans Typewriter, LucidaBright, LucidaTypewriter, New Century Schoolbook, Palatino, Rockwell, symbol, Times, Zapfchancery, and Zapfdingbats) plus some special-purpose fonts (the X11 special fonts plus XX OPEN LOOK Cursor and Glyph fonts). Hundreds of fonts are available both commercially and as "free software" for PostScript laser printers (many of these can be used in OpenWindows as well), and commercial typesetters support thousands of families.OpenWindows uses fonts in Sun's "F3" Folio format. PostScript Type Three fonts can also be used and, in revision3.1 of OpenWindows, you can use PostScript Type One as well. Both

F3 and PostScript Type One are widely available commercially. X11R5 also supports Bit-Stream's "Speedo" format fonts,, which some commercial type suppliers provide.:

Figure 10-2. xshowfonts listing for the 57 OpenWindows Fonts.

Because of the difficulty of working with fonts in X11R4, the X Consortium has included scalable fonts in X11R5, half a decade after NeWS (now part of OpenWindows) introduced them to the UNIX workstation market. X11R5 incorporates scalable font technology from BitStream, using their "Speedo" format fonts. X11R5 also includes "font server" programs that make fonts available to the X11 server. This will be especially useful to sites that run a variety of diskless X Terminals, since at present each X Terminal vendor uses a slightly different method of downloading fonts. The Font Server is described in Volume Eight, *X Window System Administrator's Guide*. However, at the present time (late 1992) few vendors are yet shipping the X11R5 release, so we still need to understand the X11R4 mechanism. By mid-1993, most reputable vendors should be supporting the X11R5 "font server" technology.

When you think of the X11 fonts as consisting of several large font families, rather than hundreds of unique fonts, you can quickly reduce the clutter. Figure 10-2 shows the major families of commercial fonts that are available under X. To illustrate the fonts, we've used the simple expedient of printing each font name in the font itself. Font names are truncated to fit on the page. (For those of you who don't read the Greek alphabet, several entries have

names like "-adobe-symbol-medium-r-normal--18..." This font is used for mathematical equations and so forth, rather than for normal display purposes.)†

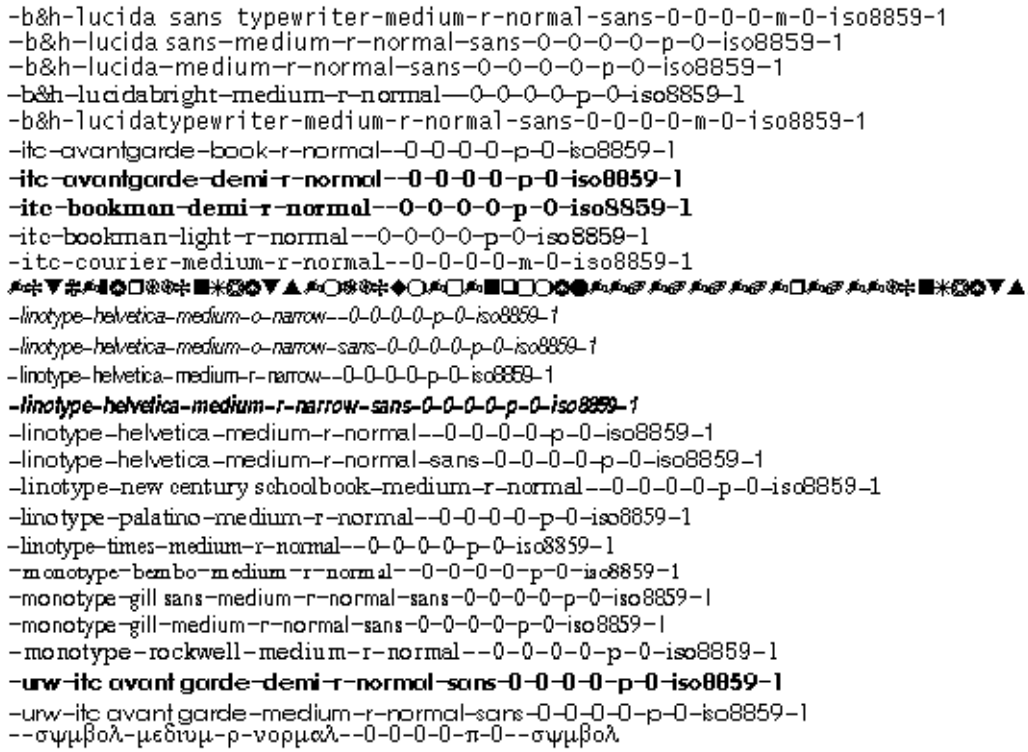


Figure 10-3. The major font families in the OpenWindows product

You'll notice that with the exception of Courier and Lucidatypewriter, all of the fonts in the figure are *proportionally spaced*. That is, each character has a separate width. This makes them look good on a printed page and for screen-aware text processing programs, but makes them less appropriate for screen display in terminal windows (especially for program editing), since text will not line up properly unless all characters are the same width.

† To generate the figures in this section and in Appendix B, *OpenWindows and X11 Fonts*, we wrote a short program called *xshowfonts*, which displays a series of fonts in a scrollable window. Source code for *xshowfonts* is listed in Appendix B, *OpenWindows and X11 Fonts*. In each case, we used wildcards (discussed later in this chapter) to select the fonts we wanted and then did screendumps of the resulting images. Note that the fonts look better on the screen than they do in the illustration, since the scaling factor used to make the screen dumps exacerbates the "jagged edges" endemic to bitmap fonts.

You will most likely use these proportional fonts for labels or menu items, rather than for running text. (Word processing or publishing programs will, of course, use them to represent proportional type destined for the printed page.)

Courier and Lucidatypewriter are *monospaced* fonts: every character has the same width. Monospaced fonts can be used effectively for the text font in *xterm* or *cmdtool* windows.

There are also some special monospaced fonts originally designed for computer displays. You can think of these as *character cell fonts*. They, too, are monospaced but the spacing relates to the size of a cell that contains each character, rather than (necessarily) to the character itself. Some of these fonts are quite old -they were originally available in X11 Release 2- and they have simple names expressing their size in pixels. For example, in the font named 8x13, each character occupies a box 8 pixels wide by 13 pixels high.They since have been renamed to use the logical font naming conventions with a foundry name of "misc," and a font family of "fixed." but the simple names have been preserved as aliases. shows the character cell fonts, using their short names

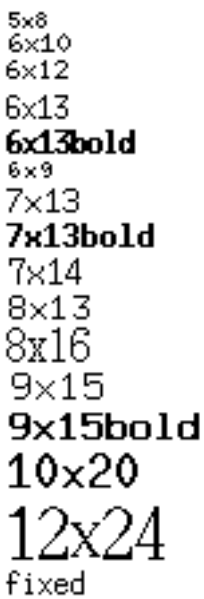


Figure 10-4. Miscellaneous fonts for xterm text

Table 10-1 shows the correspondence between these aliases and full font names. Note that the 6x13 font also has an alias called "fixed" defined for it. The "fixed" alias is used as the default font for *xterm* windows. Twelve-point Helvetica bold roman has the alias "variable," which several applications use as the default font for labels. The Motif window

manager *mwm* uses this font for the application name that appears in the titlebar. The OPEN LOOK Window Manager *olwm* uses "Lucida" for its titlebars and menus.

Table 10-1. Fixed Font Aliases and Font Names

Fixed Name	Font Name
fixed	-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
5x8	-misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-
6x9	-misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
6x10	-misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
6x12	-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
6x13	-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
6x13bold	-misc-fixed-bold-r-semicondensed--13-120-75-75-c-60-iso8859-
7x13	-misc-fixed-medium-r-normal--13-120-75-75-c-70-iso8859-1
7x13bold	-misc-fixed-bold-r-normal--13-120-75-75-c-70-iso8859-1
7x14	-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
8x13	-misc-fixed-medium-r-normal--13-120-75-75-c-80-iso8859-
8x13bold	-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
8x16	-sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-
9x15	-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-
9x15bold	-misc-fixed-bold-r-normal--15-140-75-75-c-90-iso8859-1
10x20	-misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-
12x24	-sony-fixed-medium-r-normal--24-170-100-100-c-120-iso8859-1

X11R4 also includes the Clean family of fixed-width fonts from Schumacher, and DEC's terminal fonts, both of which are illustrated in Appendix B, *OpenWindows and X11 Fonts*.

There are also many other special purpose fonts, such as the Greek Symbol font that we already saw, the cursor font, the OPEN LOOK cursor and glyph fonts, and Kana and Kanji Japanese fonts. See Appendix B, *OpenWindows and X11 Fonts*, for comprehensive lists and samples of these fonts, as well as pictures of the complete character set in some representative fonts.

10.1.2 Scalable Fonts

The X Logical Font Description described above can be used for X11R5 Speedo fonts and for OpenWindows scaleable fonts. Bitmap fonts appear just as they did above, while scal-

able fonts are listed as having a zero value for pixel size, point size, resolution, and maximum width. For example, while an X11 bitmap font like Helvetica has to be listed once for each of its limited number of point sizes, *xlsfonts* under OpenWindows or X11R5 reports each Lucida font only once. Here are the Lucida listings:

```
-b&h-lucida-bold-i-normal-sans-0-0-0-0-p-0-iso8859-1
-b&h-lucida-bold-i-sans--0-0-0-0-p-0-iso8859-1
-b&h-lucida-bold-r-normal-sans-0-0-0-0-p-0-iso8859-1
-b&h-lucida-bold-r-sans--0-0-0-0-p-0-iso8859-1
-b&h-lucida-medium-i-normal-sans-0-0-0-0-p-0-iso8859-1
-b&h-lucida-medium-i-sans--0-0-0-0-p-0-iso8859-1
-b&h-lucida-medium-r-normal-sans-0-0-0-0-p-0-iso8859-1
-b&h-lucida-medium-r-sans--0-0-0-0-p-0-iso8859-1
-b&h-lucidabright-demibold-i-normal--0-0-0-0-p-0-iso8859-1
-b&h-lucidabright-demibold-r-normal--0-0-0-0-p-0-iso8859-1
-b&h-lucidabright-medium-i-normal--0-0-0-0-p-0-iso8859-1
-b&h-lucidabright-medium-r-normal--0-0-0-0-p-0-iso8859-1
-b&h-lucidatypewriter-bold-r-normal-sans-0-0-0-0-m-0-iso8859-1
-b&h-lucidatypewriter-bold-r-sans--0-0-0-0-m-0-iso8859-1
-b&h-lucidatypewriter-medium-r-normal-sans-0-0-0-0-m-0-iso8859-1
-b&h-lucidatypewriter-medium-r-sans--0-0-0-0-m-0-iso8859-1
```

These sixteen listings provide fonts equivalent to about 96 of the bitmap fonts, and have the further advantage that they can be requested in *any* point size. For example:

```
xfd -fn '-b&h-lucida-bold-r-normal-sans--370-0-0-m-0-iso8859-1'
```

will scale the font Lucida Typewriter Bold to thirty-seven points (remember that the point sizes is in tenths, so 370 means 37 points) and display it. However, when using wildcards with this scaling, you can't be as cavalier about leaving out fields as you can with the MIT server. If in doubt, put in all the dashes, even if you're wildcarding most of the fields.

As a convenience, primarily for interactive use rather than in X Resource files, you can use the PostScript names for these fonts, followed by a dash, and the pointsize. For example,

```
xfd -fn Lucidasans-Typewriterbold-37
```

10.1.3 Stroke Weight and Slant

The characters in a given font family can be given a radically different appearance by changing the *stroke weight* or the *slant*, or both.

The most common weights are medium and bold. The most common slants are roman (upright), italic, or oblique. (Both italic and oblique are slanted; however, italic versions of a font generally have had the character shape changed to make a more pleasing effect when slanted, while oblique fonts are simply a slanted version of the upright font. In general, *serif* fonts (those with little decorations on the ends and corners of the characters) are slanted via italics, while *sans-serif* fonts are made oblique.)

Figure 10-5 compares the medium and bold weights, and the roman and italic or oblique slants in the Helvetica font family.

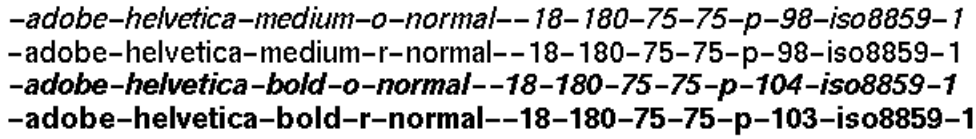


Figure 10-5. The same fonts in different weights and slants

X11 and OpenWindows also include some fonts that have an in-between weight called *demibold*. Weight names are somewhat arbitrary, since a demibold weight in one family may be almost as dark as a bold weight in another.

The font naming convention also defines two counter-clockwise slants called *reverse italic* (ri) and *reverse oblique* (ro), as well as a catch-all called *other* (ot).

10.1.4 Font Sizes

Font sizes are often given in a traditional printer's measure known as a *point*. A point is approximately one seventy-second of an inch. Most of the font families are provided in the six point sizes shown in Figure 10-6.

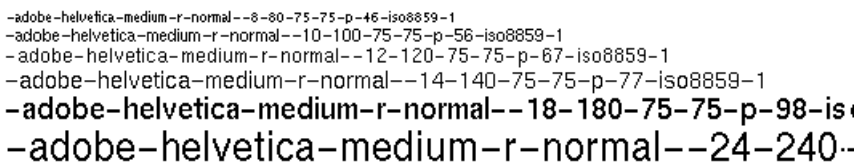


Figure 10-6. The same font in six different point sizes

However, the size story doesn't end there. Newer X servers (such as Sun's OpenWindows server and the X11R5 and later servers) support scalable outline fonts that are device-independent and, thus, true-to-size regardless of the output device. These fonts can be requested in virtually any pointsize. In older releases of X, such as X11R4, fonts we re simply bitmaps. Because of the different resolution of computer monitors, a font with a given nominal point size might actually appear larger or smaller on the screen.

Most monitors on the market today have a resolution between 75 dots per inch (dpi) and 100 dots per inch. Accordingly, there are both 75-dpi and 100-dpi versions of a few of the fonts in X11 R3, and of most of them in X11R4. These separate versions of each font are stored in different directories. By setting the font search path so that the appropriate direc-

tory comes first, you can arrange to get the correct versions without having to specify them in the font name.*†

But how do you tell which kind of monitor you have?

If you have the manufacturer's specs on your monitor, they might give you this figure. But they'll more likely give you the overall resolution in rows and columns. After measuring the physical screen, you can do some rough calculations to arrive at the equivalent in dots per inch. For example, the 16-inch monitor on the Sony NEWS workstation has an advertised resolution of 1280 x 1024 pixels. The actual viewing area is approximately 13 inches wide by 10 inches high. Dividing the resolution by the size, you come up with a vertical resolution of 102.4 dpi and a horizontal resolution of 98.5 dpi.

The Sun 19-inch monitor, by contrast, has an advertised resolution of 1152 x 900 pixels. The horizontal and vertical dimensions of the viewing area are approximately 13.75 x 10.75 inches. This yields a resolution of about 84 dpi.

What happens if you select the wrong resolution for your monitor? Given the difference in the pixel size, the same size font will appear larger or smaller than the nominal point size.

For example, consider the 75- and 100-dpi versions of the 24-point charter medium italic font:

```
-bitstream-charter-medium-i-normal--25-240-75-75-p-136-iso8859-1
-bitstream-charter-medium-i-normal--33-240-100-100-p-179-iso8859-1
```

If you look at the pixel size field, you will notice that the height of the 75-dpi version is 25 pixels, while the height of the 100-dpi version is 33 pixels. If you use the 75-dpi version on the Sun, you actually get something closer to 21.5 points (75/84*24); on a 100-dpi monitor, you will actually get something closer to 18 points (75/100*24). We noticed this right away when we first began using the Sony workstation. Because of its higher resolution, the font size we had been using on the Sun appeared much smaller.

If you are working on a lower-resolution monitor, you can take advantage of this artifact to display type as large as 32 points (the size that a 24-point 100-dpi font will appear on a 75-dpi monitor). Figure 10-7 shows the 75- and 100-dpi versions of the same 24-point font, as displayed on a Sun workstation with a 19-inch monochrome monitor. As shown, neither is actually 24 points. The 75-dpi version is actually 21.5 points, as discussed above; the 100-dpi version is about 28.5 points.*

Note that the logical font-naming convention allows for different horizontal and vertical resolution values. This would allow server manufacturers to support fonts that were

† We'll talk about how to set the font search path later in this chapter.

"tuned" for their precise screen resolution. However, the fonts shipped with the generic X11 distribution all use the same horizontal and vertical resolution.†

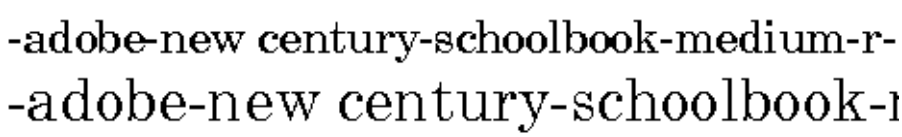


Figure 10-7. The 100-dpi version of a 24-point font appears larger on a 75-dpi monitor

As suggested above, this resolution may not exactly match the actual resolution of any particular screen, resulting in characters that are not true to their nominal point size. In the case of the Sony monitors, the actual resolution is quite close to the design of the 100-dpi fonts. However, on the Sun monitor, neither the 75- nor 100-dpi fonts will be exactly right. Unless you are running an old MIT X server that lacks scaleable fonts, however, this won't be much of a problem.)

10.1.5 Other Information in the Font Name

What we've already shown summarizes the most important information in the font name. The remaining fields are explained below:

Foundry Font manufacturers are still referred to as foundries, from the days when type was founded, or cast, from lead. The X font naming convention specifies the foundry as the company that digitized or last modified the font, rather than its original creator. For the fonts contained in the standard X distribution, the foundry is not terribly significant since there are no cases where the same font family is available from different foundries. However, there are numerous commercial font families available from more than one foundry. In general, the appearance of the fonts should be quite similar since the font family defines the design of the typeface. However, there may be some small differences in the quality of some of the characters, and there may be more significant differences in the font metrics (the vertical or horizontal measurements of the characters). This might be significant for a publishing application that was using the bitmapped font for a *wysiwyg** screen display that needed to match the fonts in a particular laser printer or typesetter.‡

"Set width" A value describing a font's proportionate width, according to the foundry. Typical set widths include: normal, condensed, semicondensed, narrow, double

† Note that the differences are exaggerated further in printing the screen dump of this display, since we scaled the bitmap up somewhat to print it..

‡ This is an acronym for "what you see is what you get" and describes a type of text editor or word processor that purports to display the page exactly as it would appear in print. There is always some approximation due to the differences between screen fonts and printer fonts. MacWrite® is a *wysiwyg* program in the MacIntosh world; FrameMaker, Interleaf and others offer *wysiwyg* programs in the X11 world.

width. All of the X11 Release 3 fonts and most of the Release 4 fonts have the set width *normal*. A few of the Release 4 fonts have the set width *semicondensed*.

Spacing All standard X11 Release 3 fonts are either m (monospace, i.e., fixed-width) or p (proportional, i.e., variable-width). In Release 4, fonts may also have the spacing characteristic c (character cell, a fixed-width font based on the traditional typewriter model, in which each character can be thought to take up the space of a "box" of the same height and width). As mentioned earlier, the original R2 fonts were this type.

Average width Mean width of all characters in the font, measured in tenths of a pixel. You'll notice, if you look back at Figure 10-2, that two fonts with the same point size (such as New Century Schoolbook and Times) can have a very different average character width. This field can sometimes be useful if you are looking for a font that is especially wide or especially narrow. The Schumacher Clean family of fonts offers several fonts in the same point size but with different average widths.†

Character set In the initial illustration of the font naming convention, Figure 10-1, we identified the character set as a single field. If you look more closely, you'll realize it is actually two fields, the first of which identifies the organization or standard registering the character set, the second of which identifies the actual character set.

Most fonts in the standard X distribution contain the string "iso8859-1" in their names, which represents the ISO Latin-1 character set. The ISO Latin-1 character set is a superset of the standard ASCII character set, which includes various special characters used in European languages other than English. See Appendix H of Volume Two, *Xlib Reference Manual*, for a complete listing of the characters in the ISO Latin-1 character set.

Note, however, that the symbol font contains the strings "adobe-fontspecific" in this position. This means that Adobe Systems defined the character set in this font, and that it is font-specific. You can see from this example that the use of these fields is somewhat arbitrary.

Style Not represented in the example or in most R3 or R4 font names. However, according to the logical font convention, the style of a font may be specified in the field between set width and pixels. Some of the possible styles are *i* (informal), *r* (roman), *serif* and *sans* (serif). Note that the *r* for roman may also be used in the slant field.

If you need still more detail and want a complete technical description of font naming conventions, see the X Consortium Standard, *X Logical Font Description Conventions*. This document is available as part of the standard MIT X distribution, and is reprinted as an Appendix in Volume Zero, *X Protocol Reference Manual*.

† These fonts all (incorrectly to our minds) have a set width of "normal." They should be distinguished by set widths such as condensed, semi-condensed, etc. Since they do not, they can be distinguished by the difference in their average width.

10.2 Font Name Wildcarding

In order to simplify font specification, X recognizes wildcarding within font names. You can use wildcarded font names to specify the text font for a client, either on the command line or in an X resource specification. You can also supply a wildcarded font name to any of the font utilities, such as *xlsfonts*, *xfd*, etc.

An asterisk (*) can be used to represent any part of the font name string; a question mark (?) can be used to represent any single character. You can usually get the font you want by specifying only the font family, the weight, the slant, and the point size, and wildcarding the rest. For example, to get Courier bold at 14 points, you could use the command line option:

```
-fn '*courier-bold-r*140*'
```

That's starting to seem a little more intuitive!

However, there are a number of "gotchas."

- First, since the UNIX shell also has a special meaning for the "&", "*" and "?" wildcard characters, wildcarded font names must be quoted. This can be done by enclosing the entire font name in quotes (as in the previous example), or by "quoting" each wildcard character by typing a backslash before it. (If you don't do this, the shell will try to expand the * to match any filenames in the current directory and, if you use the C shell, will give the message "No match." and not run the command) Wildcards need not be quoted in resource files.
- Second, if the wildcarded font name matches more than one font, the server will use the first one that matches. And unfortunately, because the font names may differ from one server to the next and are sorted in simple alphabetical order, it is indeterminate what font you will get. On X11R4, since the bold weight sorts before medium, and italic and oblique slants before roman, the specification:

```
-fn '*courier*'
```

gave Courier bold oblique, rather than the Courier medium roman you might intuitively expect. On OpenWindows 3.1, it yields a fairly small, but medium roman, Courier:

```
% xlsfonts '*courier*' | head -1
-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
%
```

As shown, you can use *xlsfonts ... | head -1* to find out what font is matched. If you aren't sure whether your wildcarded name is specific enough, try using it as an argument to *xlsfonts*. If you get more than one font name as output, you may not get what you want. Try again with a more specific name string.

The exception to this rule has to do with the *75dpi* and *100dpi* directories. If a wildcard matches otherwise identical fonts in these two directories, the server will actually use the one in the directory that comes first in the font path. This means that you should put the appropriate directory first in the font path. (We'll tell you how to do this in the next section.) Thereafter, you can generally wildcard the resolution fields (unless you specifically want a font from the directory later in the path).†

Third, the * wildcard expansion is resolved by a simple string comparison. So, for example, if you type:

```
-fn '*courier-bold*r*140*'
```

instead of:

```
-fn '*courier-bold-r*140*'
```

(the difference being the asterisk instead of the hyphen before the "r" in the slant field), the "r" would also match the "r" in the string "normal" in the set width field. The result is that you would select all slants. Since o (oblique) comes before r (roman), and you always get the first font that matches, you'd end up with Courier oblique.

The trick is to be sure to include at least one of the hyphens to set the -r- off as a separate field rather than as part of another string.

Even though a wildcarded name such as:

```
*cour*b*r-*140*
```

should get you 14-point Courier bold roman, we think it is good practice to spell out the font family and weight and use hyphens between adjacent fields. As usual there are exceptions: the Lucida family really has three subfamilies; you can get all three by specifying the family as "Lucida*" rather than "Lucida-"; and you might certainly want to abbreviate "New Century Schoolbook" to "New Century*" or "*Schoolbook."

Font names are case-insensitive. "Courier" is the same as "courier."

✗ Some early versions of the SCO "Open DeskTop" server fail to be case-insensitive in fontnames.

Table 10-2 summarizes the values you can use to specify a unique font name (assuming only the standard fonts are loaded). Choose one element from each column. Don't forget to include the leading and trailing asterisks and the hyphen before the slant.

Table 10-2. Essential Elements of a Font Name

*	Family	-	Weight	-	Slant*	-	Point Size	*
	Charter		Medium		r (roman)		80 (8 pt.)	
	Courier		Bold		i (italic)		100 (10 pt.)	
	Helvetica		Demibold		o (oblique)		120 (12 pt.)	

† Unlike *xfontsel*, which displays fonts in the order of wildcard matches, *xlsfonts* will always list fonts in straight-sort order, with the sort done character by character across the line. Since size in pixels comes before point size in the name, and the size in pixels of the 100-dpi fonts is larger than that of the equivalent 75-dpi font, the 75-dpi font will always be listed first for a given point size. But when listing more than one point size, the fonts will be jumbled. For example, the size in pixels of the 8-point Charter font at 100-dpi is 11, so it will come after the 10-point Charter font at 75 dpi, with a size in pixels of 10. The 8-point Charter font at 75 dpi gets sorted to the very end of the list, since to a character-by-character sort, its size in pixels (8) looks larger to the size in pixels of even the largest 100-dpi font (the 24 point, with a height of 33 pixels).

Table 10-2. Essential Elements of a Font Name

*	Family	-	Weight	-	Slant*	-	Point Size	*
	New century				ri (reverse italic)		140 (14 pt.)	
	schoolbook							
	Symbol				ro (reverse oblique)		180 (18 pt.)	
	Times				ot (other)		240 (24 pt.)	
	Fixed (R4)							
	Clean (R4)							
	OPEN LOOK (R4)							
	Lucida (R4)							
	Terminal (R4)							

10.3 The Font Search Path

The OpenWindows Release 3.0 server uses the directories shown in Table 10-5 to store fonts..

Table 10-3. Standard Font Directories, OpenWindows Release 3.0

Directory	Contents
/usr/openwin/lib/fonts	Main body of fonts, mostly outline
/usr/openwin/lib/fonts/100dpi	Fixed- and variable-width fonts, 100 dpi (all font families).
/usr/openwin/lib/fonts/afm	Adobe Font Metrics
/usr/openwin/lib/fonts/map	Contains one file, *fontmaps.ps*.

The OpenWindows server uses only the first of these directories as its font search path. The normal font directory is *$OPENWINHOME/lib/fonts*, and this is the default setting of the environment variable FONTPATH. FONTPATH is a colon-separated list of directories, like the normal UNIX environment PATH. For example, to add a local font directory, you

might set FONTPATH to, the following (and `export` it if using the Bourne or Korn shell):

```
/usr/openwin/lib/fonts:/usr/local/lib/fonts
```

Then if you re-start the OpenWindows server, any fonts it recognizes in the local font directory will automatically be available.

The OpenWindows 3.3 server, by contrast, uses the following font directories:

Table 10-4. Standard Font Directories,OpenWindows 3.3

Directory	Contents
/usr/openwin/lib/X11/fonts/F3/	Folio Format scaleable fonts
/usr/openwin/lib/X11/fonts/F3bitmaps/	Folio Format pre-scaled fonts
/usr/openwin/lib/X11/fonts/Type1/	PostScript Type 1 scaleable fonts
/usr/openwin/lib/X11/fonts/Speedo/	Speedo format scaleable fonts
/usr/openwin/lib/X11/fonts/misc/	Same as X11 misc fonts
/usr/openwin/lib/X11/fonts/75dpi/	Many 75dpi fonts
/usr/openwin/lib/X11/fonts/100dpi/	Many 100dpi fonts
/usr/openwin/lib/X11/fonts/Xt+/	Glyph fonts for OLIT toolkit

In the MIT X server and its derivatives, fonts are stored in three directories, as shown in Table 10-5.

Table 10-5. Standard Font Directories, X11 Release 4

Directory	**Contents**
/usr/lib/X11/fonts/misc	Sixty fixed-width fonts, including the six available in Release 3, the cursor font, several Clean family fonts provided by Schumacher, a Kanji font, Kana fonts, and OPEN LOOK cursor and glyph fonts.
/usr/lib/X11/fonts/75dpi	Fixed- and variable-width fonts, 75 dpi.
/usr/lib/X11/fonts/100dpi	Fixed- and variable-width fonts, 100 dpi (all font families).

These three directories (in this order) constitute X’s default font path.

If you wish to provide additional fonts, other directories can be added to the font search path, or its order can be rearranged, using *xset* with the `fp` option. To completely replace the font path, simply specify a comma-separated list of directories. For example, to put the *100dpi* directory before the *75dpi* directory with the MIT server, you might enter:

```
% xset fp=
          /usr/lib/X11/fonts/misc,/usr/lib/X11/fonts/100dpi
          ,\ /usr/lib/X11/fonts/75dpi
```

(Note that a space must follow the equal sign (=) and that the example above is broken onto two lines escaped with a backslash (\) only so that it can be printed within the page margins.) To restore the default font path, type:

```
% xset fp default
```

Use the `fp+` option to add a directory or list of directories to the end of the font path, or `+fp` to add them at the start. Use `-fp` and `fp-` to delete directories from the beginning or end of the font path.

For a complete list of the fonts in each directory and samples of each font, refer to Appendix B, *OpenWindows and X11 Fonts*.

10.3.1 The fonts.dir Files (Standard X server)

In addition to font files, the MIT X server requires each font directory to contain a file called *fonts.dir*. The *fonts.dir* files serve, in effect, as databases for the X server. When the X server searches the directories in the default font path, it uses the *fonts.dir* files to locate the font(s) it needs.

Each *fonts.dir* file contains a list of all the font files in the directory with their associated font names in two-column form. (The first column lists the font file name and the second column lists the actual font name associated with the file.) The first line in *fonts.dir* lists the number of entries in the file (i.e., the number of fonts in the directory).

Example 10-1 shows a portion of the *fonts.dir* file from the Release 4 */usr/lib/X11/fonts/100dpi* directory. As the first line indicates, the directory contains 200 fonts. The first group of fonts listed below (up to the second ellipse) are available as of X11 Release 4. They are all Courier family fonts. (These fonts are 100-dpi equivalents of fonts that were only available in 75 dpi in X11 Release 3.) The second group of fonts shown in the list below (a few sizes from the Charter family) are also available in the X11 Release 3 *100dpi* directory.

```
200
courBO08.snf -adobe-courier-bold-o-normal--11-80-100-100-m-60-
iso8859-1
courBO10.snf -adobe-courier-bold-o-normal--14-100-100-100-m-90-
iso8859-1
courBO12.snf -adobe-courier-bold-o-normal--17-120-100-100-m-100-
iso8859-1
courBO14.snf -adobe-courier-bold-o-normal--20-140-100-100-m-110-
iso8859-1 c
ourBO18.snf -adobe-courier-bold-o-normal--25-180-100-100-m-150-
iso8859-1
```

```
courBO24.snf -adobe-courier-bold-o-normal--34-240-100-100-m-200-
iso8859-1
courB08.snf -adobe-courier-bold-r-normal--11-80-100-100-m-60-iso8859-
1
courB10.snf -adobe-courier-bold-r-normal--14-100-100-100-m-90-
iso8859-1
courB12.snf -adobe-courier-bold-r-normal--17-120-100-100-m-100-
iso8859-1
courB14.snf -adobe-courier-bold-r-normal--20-140-100-100-m-110-
iso8859-1
courB18.snf -adobe-courier-bold-r-normal--25-180-100-100-m-150-
iso8859-1
courB24.snf -adobe-courier-bold-r-normal--34-240-100-100-m-200-
iso8859-1
courO08.snf -adobe-courier-medium-o-normal--11-80-100-100-m-60-
iso8859-1
courO10.snf -adobe-courier-medium-o-normal--14-100-100-100-m-90-
iso8859-1
courO12.snf -adobe-courier-medium-o-normal--17-120-100-100-m-100-
iso8859-1
courO14.snf -adobe-courier-medium-o-normal--20-140-100-100-m-110-
iso8859-1
. . .
```

Example 10-1Subsection of the Release 4 fonts.dir file in /usr/lib/X11/fonts/100dpi

The *fonts.dir* files are created by the *mkfontdir* client when X is installed. *mkfontdir* reads the font files in directories in the font path, extracts the font names, and creates a *fonts.dir* file in each directory. If *fonts.dir* files are present on your system, you probably won't have to deal with them or with *mkfontdir* at all. If the files are not present, or if you have to load new fonts or remove existing ones, you will have to create files with *mkfontdir*. Refer to Volume Eight, *X Window System Administrator's Guide*, for details.

10.4 Font Name Aliasing

Another way to abbreviate font names is by aliasing (that is, by associating fonts with alternative names of your own choosing).

10.4.1 Aliases—X11R5 and OpenWindows 3.3 Server

You can edit or create a file called *fonts.alias*, in any directory (or multiple directories) in the font search path, to set aliases for existing fonts. The X server uses both *fonts.dir* files and *fonts.alias* files to locate fonts in the font path.

If you are running X11 , there should already be an alias file in each font directory. Take the time to look at the contents of each of these files, since many of the existing aliases may be easier to type than even wildcarded font names. You can also add aliases to the file, change existing aliases, or even replace the entire file. However, this should be done with caution. To play it safe, it's probably a good idea merely to *add* to existing *fonts.alias* files. If you're working in a multiuser environment, the system administrator should definitely be consulted before aliases are added or changed. Note that when you create or edit a

fonts.alias file, the server does not *automatically* recognize the aliases in question. You must make the server aware of newly created or edited alias files by resetting the font path with *xset*.

The *fonts.alias* file has a two-column format similar to the *fonts.dir* file: the first column contains aliases, the second contains the actual font names. If you want to specify an alias that contains spaces, enclose the alias in double quotes. If you want to include double quotes (") or other special characters as part of an alias, precede each special symbol with a backslash ("\".

When you use an alias to specify a font in a command line, the server searches for the font associated with that alias in every directory in the font path. Therefore, a *fonts.alias* file in one directory can set aliases for fonts in other directories as well. You might choose to create a single aliases file in one directory of the font path to set aliases for the most commonly used fonts in all the directories. Example 10-2 shows three sample entries that could be added to an existing *fonts.alias* file (or constitute a new one).

```
xterm12-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
xterm14-adobe-courier-medium-r-normal--14-140-75-75-m-90-iso8859-1
xterm18-adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1
```

Example 10-2. Sample fonts.alias file entries

As the names of the aliases suggest, these sample entries provide aliases for three fonts (of different point sizes) that are easily readable in *xterm* windows. (We also recommend the fixed-width font stored in the file *9x15.snf,** in the *misc* directory.) You can also use wildcards within the font names in the right column of an alias file. For instance, the alias file entries above might also be written as follows:

```
xterm12*courier-medium-r-*-120*
xterm14*courier-medium-r-*-140*
xterm18*courier-medium-r-*-180*
```

Once the server is made aware of aliases, you can specify an alias on the command line. For example, you can use a font name alias as an argument to *xfd*.

A special note about the *misc* directory: when X was configured for your system, a *fonts.alias* file should have been created in this directory. The first two entries in this file are shown below:

```
fixed -misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
 variable -*-helvetica-bold-r-normal-*-*-120-*-*-*-*-*-*
```

The default file contains an additional 56 entries but the entries pictured above are particularly important. The aliases called "fixed" and "variable" are invoked as the default fonts for many clients. The "fixed" font can be thought of as a system-wide default. The "variable" font, described in the right column as a 12-point bold Helvetica font, is used as the default font by *bitmap*, as well as by other clients. If this file is removed or replaced, when you run *bitmap*, you'll get an error message that the server cannot open the variable font, and text in the *bitmap* window will display in the smaller, somewhat less readable "fixed" font.

If you do choose to edit the *fonts.alias* file in the *misc* directory, it is important to preserve at least these two aliases. (As we've said, it's probably a better idea to keep all the default entries and merely append any new ones.)

If you're running the older X11 Release 3, the *fonts.alias* file in the *misc* directory will be somewhat different. The X11 Release 3 version of the *fonts.alias* file in the *misc* directory comprises only these two lines:

```
fixed 6x13 variable *-helvetica-bold-r-normal-*-*-140-*
```

Regardless of what edits you make to the file, the line specifying the variable alias must not be changed.

The variable font is slightly larger in X11 Release 3 (14-point) than in X11 Release 4 (12-point). If you examine the Release 3 alias file a little more closely, you may notice that the first line contains an *incorrect* alias specification. Remember, in Release 3, *fixed* is actually the name of the default system font—it is not an alias. The first column should contain aliases; the second column should contain proper font names. However, 6x13 is not a proper font name. It is actually the name of the file that contains the font named *fixed*. You can specify *fixed* as a font on the command line and it will work—but as a font name, not an alias.

10.4.2 Aliases—Older OpenWindows XNews Server

You can provide your own aliases by creating a *.user.ps* file with the following contents. This example makes "Nice" be an alias for the existing Folio font "GillSans":

```
FontDirectory begin
    /Nice/Gill-Sans_FontDirectorySYN
    % other aliases here
end
```

The font Nice can now be used instead of Gill-Sans at any point size. As usual, you must re-start the server for changes in this file to take effect. This technique is in fact used to provide X11 names for the standard PostScript fonts. The file *$OPENWINHOME/lib/fonts/Synonyms.Lst* is comprised of hundreds of lines like this:

```
/-bitstream-charter-medium-i-normal--8-80-75-75-p-44-iso8859-1 \
    /Charter-Italic _FontDirectorySYN
```

See the Sun document *X11/NeWS Version 2 Server Guide*, part number 800-4898-10, for more details on the user's initialization files.

10.4.3 Making the Server Aware of Aliases

After you create (or update) an alias file, the server does not automatically recognize the aliases in question. You must make the server aware of newly created or edited alias files by "rehashing" the font path with *xset*. Enter:

```
% xset fp rehash
```

on the command line. The *xset* option `fp` (font path) with the `rehash` argument causes the server to reread the *fonts.dir* and *fonts.alias* files in the current font path. You need to do this every time you edit an alias file. (You also need to use *xset* if you add or remove fonts. See Appendix B, *OpenWindows and X11 Fonts*, for details.)

10.5 Utilities for Displaying Information about Fonts

We've already mentioned *xlsfonts*, which simply displays the names and aliases of available fonts. In addition, *xfd* can be used to display the full character set of a particular font, and *xfontsel* can be used to interactively preview and select a font for use in another window.

10.5.1 The Font Displayer: xfd

If you're unfamiliar with the general appearance of a particular font, we've included pictures of some representative fonts in Appendix B, *OpenWindows and X11 Fonts*.

You can also display the characters in a font using the *xfd* (font displayer) client. Note that *xfd* takes an option, `-fn`, before the font name. For example, to display the default system font, a 6x13 pixel fixed-width font known as *fixed*, enter:†

```
% xfd -fn fixed &
```

The *xfd* window will display the specified font (by its real name; *fixed* is an alias) as shown in Figure 10-8.

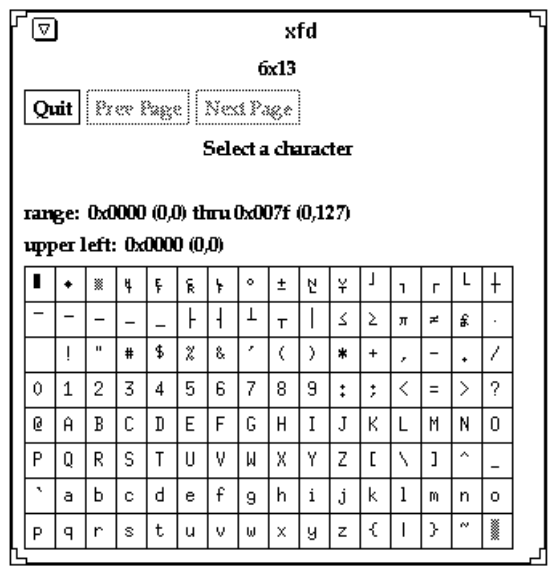


Figure 10-8. Fixed font, 6x13 pixels

† In X11 Release 3, *fixed* is a font name. In Release 4, it is an alias for a longer font name that follows the conventions outlined previously.

This figure depicts the X11 Release 4 version of *xfd*. The font name is displayed across the top of the window Three command buttons appear in the upper-left corner of the window below the font name. If the font being displayed doesn't fit within a single *xfd* screen, Prev Page and Next Page allow you to scroll through multiple screens. (The horizontal and vertical window dimensions can vary slightly to accommodate different fonts but certain fonts will still require multiple screens.) The Quit button causes the application to exit, though this can also be done by typing q or Q while input is focused on the *xfd* window.

In addition to displaying a font, *xfd* also allows you to display certain information about the individual characters. But before we examine these capabilities, let's take a closer look at the way the characters in a font are identified and how the *xfd* window makes use of this information.

Within a font, each character is considered to be numbered. The *xfd* client displays a font's characters in a grid. By default, the first character of the font appears in the upper-left position; this is character number 0. The two text lines above the grid identify the upper-left character and the range of characters in the window by character numbers both in hexadecimal and in decimal notation (in parentheses following the hex character number).

You can specify a character other than character number 0 to be in the first position in the window using the `-start` option. For example, if you enter this command line:

```
% xfd -start 15 -fn fixed &
```

the *xfd* window begins with character number 15.

Notice the instruction `Select a character` below the command buttons. To display information about a particular character, click any pointer button within the character's grid square. Statistics about the character's width, left bearing, right bearing, ascent, and descent are displayed where the line `Select a character` previously appeared.

The *xfd* client is most useful when you have an idea what font you might want to display. If you don't have a particular font in mind or would like to survey the possibilities, the *xfontsel* client (available as of X11 Release 4) allows you to preview a variety of fonts by specifying each component of the font name using a different menu.

10.5.2 Previewing and Selecting Fonts: xfontsel

The *xfontsel* client provides a font previewer window in which you select the font to view using 14 menus corresponding to the 14 components of a font name. By specifying various font name components, you can take a look at a variety of fonts. This is particularly useful if you are trying to pick good display fonts and you don't have a clear idea what type of font would be best. Rather than running several instances of *xfd*, you can dynamically change the font displayed in the *xfontsel* window by changing the font name components. (Despite the flexibility of *xfontsel*, it's certainly not practical to preview *all* of the available fonts. If you have no idea what a particular font family looks like, see the discussion earlier in this chapter, or refer to Appendix B, *OpenWindows and X11 Fonts*, for complete listings.)†

Once you've displayed the desired font using the menus, you can make the name of that font the PRIMARY text selection by clicking on the window's select button. You can then

paste the font name into another window using the pointer: onto a command line, into a resource file, etc. Making a font name the PRIMARY selection also enables you to choose that font from the *xterm* VT Fonts menu. (Selecting text and using *xterm* menus are described in Chapter 5, *The xterm Terminal Emulator*.

10.5.2.1 Previewing Fonts with the xfontsel Menus

To run *xfontsel*, enter this command in an *xterm* window:

```
% xfontsel &
```

The *xfontsel* window initially displays a some randomly-chosen font such as a greek symbol font or a bold, constant-width, 7x13 pixel font, as shown in Figure 10-9. In fact it displays the first font in the default font search path.

Figure 10-9. xfontsel window displaying random font

The upper-left corner of the *xfontsel* window features two command buttons: *quit* and *select*. As we've explained, clicking on *select* with the first pointer button makes the font displayed in the window the PRIMARY text selection; obviously, *quit* causes the application to exit.

Below the command buttons is, in effect, a generic font name or font name template. It is divided into 14 fields corresponding to the 14 parts of a standard font name. Each field is an abbreviation for one part of a font name. Take a look again at the sample font name in Figure 10-1 to refresh your memory as to the components. Each of the fields in the *xfontsel* window is actually the handle to a menu which lets you specify this part of the font name.

To get a clearer idea of how this works, move the pointer onto the generic font name—specifically onto the first field, fndry. (This is an abbreviation for the first part of a font name, the foundry.) When you place the pointer on fndry, the field title should be highlighted by

† To our minds, the major drawback of *xfontsel* is that it shows you only the first font that matches a given wildcarded font name. A far better interface would list all of the matching fonts so that you could compare and choose the one that best suited your needs. There is no way in the standard X distribution to display the appearance of a group of fonts. To produce the figures in this book, we had to write such a program, which we called *xshowfonts*. The program has since been posted to *comp.sources.x*, and a listing appears in Appendix B, *OpenWindows and X11 Fonts*.

a box. You can then display a menu of foundry names by pressing and holding down the first pointer button, as in Figure 10-10

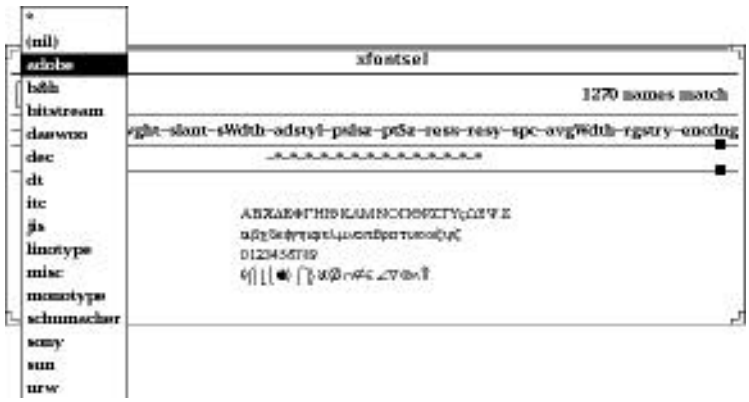


Figure 10-10. xfontsel window with foundry menu displayed

Notice that the first choice is the asterisk (*) wildcard character. This is the first choice on all of the menus, allowing you to include wildcards in the font name you specify rather than explicitly selecting something from all 14 menus.

To specify a font name component (i.e., make a selection from the menu), first display the menu by pressing and holding down the first pointer button. Then move the pointer down the menu. As the pointer rests on each menu item, it is highlighted by reverse video. To select a highlighted menu item, release the first pointer button.

The line below the font name menus represents the actual font name. When you first run *xfontsel*, all of these fields contain wildcard characters because no menu selections have been made. The number of fonts matched by the font name is displayed in the upper-right corner of the window. The number of fonts initially matched depends on the number of fonts with this naming convention available on your system. In this example, 474 fonts match. (Since this line of wildcards can match *any* 14-part font name, the server chooses the first font in the font path that reflects this naming convention.)

When you select a font name component from one of the 14 menus, the component appears in the actual font name, and the *xfontsel* window displays the first font that matches this

name. For example, say we select adobe from the fndry menu, the *xfontsel* window would look like Figure 10-11.

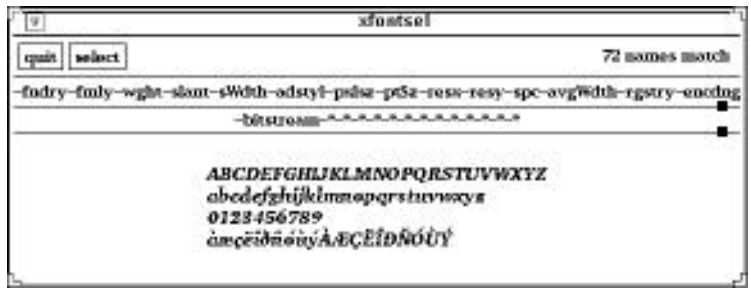


Figure 10-11. xfontsel after choosing Bitstream from the foundry menu

The font name is now:

```
-bitstream-*-*-*-*-*-*-*-*-*-*-*-*-*
```

and the window displays the first font in the font path that matches this wildcarded name. In this case, the first font to match is a 12-point bold Oblique Courier font, which is stored in the file *courBO10.snf* and has the actual font name:

```
-bitstream-charter-bold-o-normal-*-10-100-75-75-m-60-iso8859-1
```

Once you make a selection from one menu, the number of possible fonts matched by the name changes. (Notice the line `nnn fonts match` in the upper-right corner of the window.) Choosing one font name component also eliminates certain choices on other menus. For example, after you select Bitstream as the foundry, the possible choices for font family (the second menu, fmly) are narrowed down to 2 (not counting the asterisk). Again display the fmly menu using the first pointer button. The available choices for font family appear in a regular typeface; the items that are unavailable (i.e., cannot be selected) appear in a lighter typeface. Families such as Clean, Lucida, and Charter are in a lighter typeface because none of the standard X fonts provided by Bitstream is from these families. Bitstream fonts in the standard X distribution are limited to the families Courier, and Charter; these are the items available on the fmly menu.

In order to display a particular font, you'll probably have to make selections from several of the menus. As described earlier in the section "Font Name Wildcarding," we suggest you explicitly select at least these parts of the font name:

- Font family
- Weight
- Slant
- Point size

Thus, you would make selections from the fmly, wght, slant, and ptSz menus.

You can also use the *-pattern* option with a wildcarded font name to start out with a more limited range of options. For example, if you typed:

```
% xfontsel -pattern '*charter-bold-o-*'
```

you'd start out with the pattern you specified in the filename template part of the *xfontsel* display. You could then simply select from the ptSz menu to compare the various point sizes of Courier bold oblique until you found the one you wanted.

Note that if the pattern you specify to *xfontsel* matches more than one font, the one that is displayed (the first match found) is the one that the server will use. This is in contrast to *xlsfonts*, which sorts the font names. You can always rely on *xfontsel* to show you the actual font that will be chosen, given any wildcard specification.

10.5.2.2 Selecting a Font Name

Once you make selections from the menus to compose the name of the font you want, the corresponding font is displayed in the *xfontsel* window. Then you can select that font name by clicking on the select command button with the first pointer button. The font name becomes the PRIMARY text selection and thus can be pasted in another window using the second pointer button, as described in Appendix A, *The xterm/olterm Terminal Emulator*.

You might paste the font name on a client command line in an *xterm* window in order to specify it as the client's display font. (See Chapter 11, *Command Line Options*.) You might paste it into a resource file such as *.Xdefaults* to specify it as the default font for a client or some feature of a client (such as a menu). (See Chapter 12, *Setting Resources*, for more information.)

Less obviously, once a font name is made the PRIMARY text selection, it can be toggled as the *xterm* display font using the Selection item of the *xterm* VT Fonts menu. The Selection menu item can only be chosen from the VT Fonts menu when there is a PRIMARY text selection. (Otherwise, the menu item appears in a lighter typeface, indicating that it is not available.) If the PRIMARY text selection is a valid font name (as it is when you've pressed the select button in the *xfontsel* window), the *xterm* window displays in that font. (In cases where the PRIMARY selection is not a valid font name, the *xterm* display font does not change.)

By default, *xfontsel* displays the lowercase and uppercase letters a through z and the digits 0 through 9. You can specify alternative sample text using the `-sample` option. For more information about this and other options, see the *xfontsel* reference page in Part Three of this guide.

10.5.3 The "text" demo program (OpenWindows up to 3.2)

OpenWindows Release Two includes a useful program for displaying text in various fonts. The *text* program is stored in the demos directory, which is usually not in your standard directory search path. It is a PostScript script file, so you can invoke it using *psh* as follows:

```
% cd $OPENWINHOME/demos
% psh ./text
```

The initial screen (Figure 10-12)shows some silly text in the default typeface:

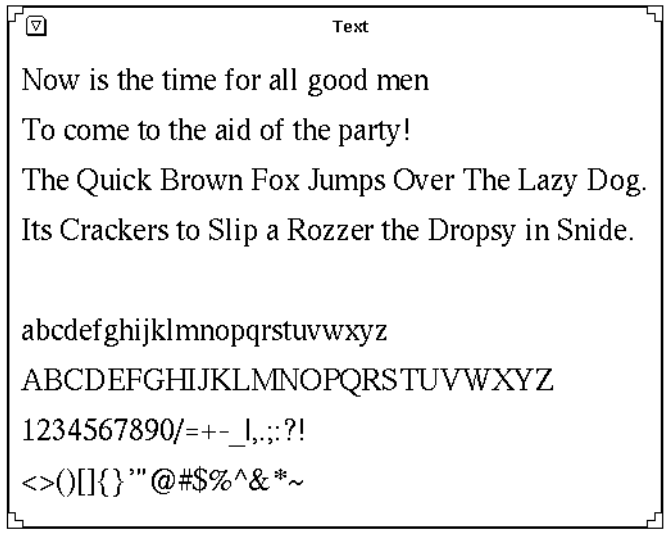


Figure 10-12. Text Demo.

The MENU button in the program's window displays this menu: l. Fonts Point Size Gray Level Contrast Show Text Show All The last two are used to toggle between the default display above, and the display of the full character set in any given font, as shown in Figure 10-13.

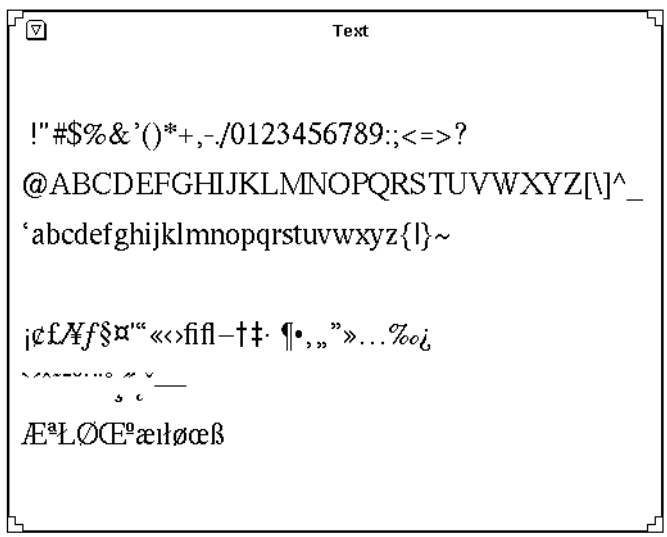


Figure 10-13. Text Demo.

The first item in the menu, *fonts*, expands to a (very large) list of all the available font names. Note that it takes quite a while to prepare this menu, as the X/NeWS server has to examine each font file at its disposal. Figure 10-14 is the list on my system:

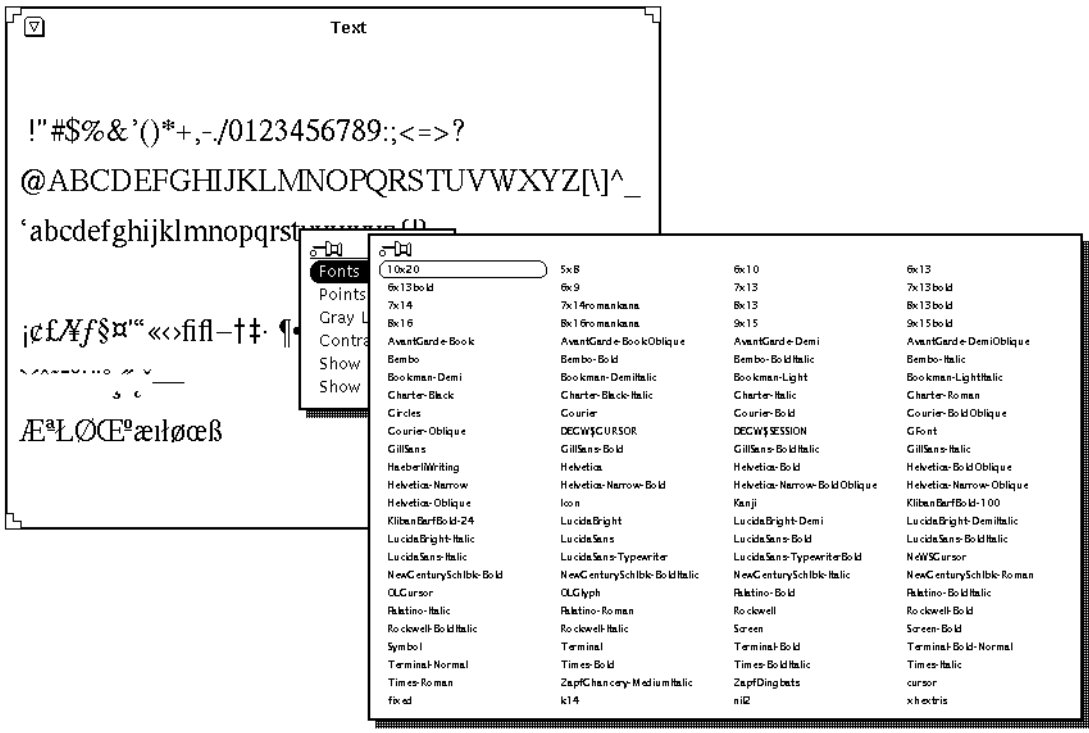


Figure 10-14. ***Text*** Demo: List of Fonts

If you click on a large font, you will only see the first part of it; use *xfd* mentioned above to see it all. For example, if you click on *Kanji*, you see a window like Figure 10-15.

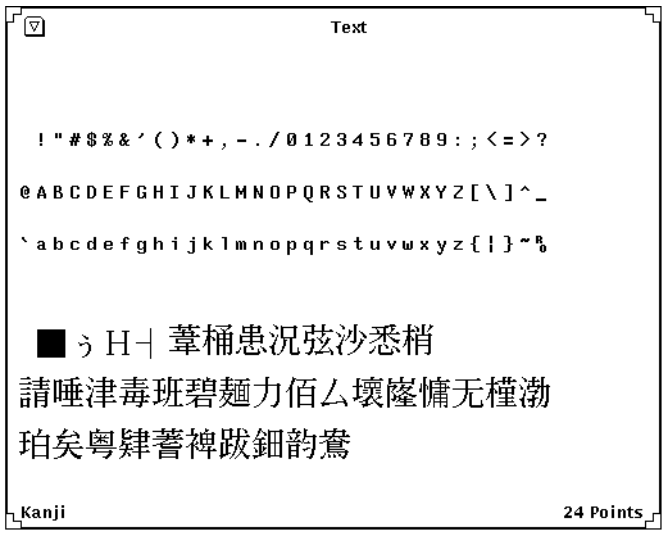


Figure 10-15. Text demo: Kanji Font (first page of many).

Note that the font menu is pinnable; if you are planning to experiment with a variety of fonts, you can (and should) pin this menu up.

The *Point Size* menu lets you view the text sample at a variety of sizes. The remaining items, *Gray Level* and *Contrast* allow you to view the text in a variety of shadings and contrasts, including reverse video.

All in all, *text* is a useful tool for previewing all the available fonts in the OpenWindows system.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 11

Command Line Options

This chapter describes command line options that are common to most clients. Some arguments to command line options can also be specified as the values of resource variables, described in Chapter 12, *Setting Resources*. For example, the format of a geometry string or a color specification is the same whether it is specified as an argument to an option or as the value of a resource definition.

As explained in Chapter 3, *Opening Additional Windows*, The X Window System allows the user to specify numerous (very numerous!) command line options when starting most clients. In order to know for sure what options a given client will accept, you need to check its reference page in Part Three. However, there are many "generic" options. In addition to certain client-specific options, all applications built with the MIT X Toolkit (or a toolkit based on the Xt Intrinsics, such as OLIT, the OPEN LOOK Intrinsics Toolkit, or OSF/Motif) accept certain standard options, which are listed in Table 11-1. Programs built using the XView toolkit accept most of these options, as well as certain XView-specific options and some "-W" short forms shown in Table 11-2, "XView Generic Options". Some non-Toolkit applications may also recognize some or all of these options. The first column contains the name of the option, the second the name of the resource to which it corresponds (see Chapter 12, *Setting Resources*), and the third a brief description of what the option does.

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

Table 11-1. MIT Toolkit and OLIT Standard Options

Flag	Long Flag	Resource	Description
`-bg`	`-background`	background	Background color of window.
`-bd`	`-bordercolor`	borderColor	Color of window border.

Table 11-1. MIT Toolkit and OLIT Standard Options

Flag	Long Flag	Resource	Description
`-bw`	`-borderwidth`	borderWidth	Border width of window in pixels.
	`-display`	display	Display on which to run client.
`-fn`	`-font`	font	Font for text display.
`-fg`	`-foreground`	foreground	Foreground (drawing or text) color of window.
	`-geometry`	geometry	Geometry string for window size and placement.
	`-iconic`		Start the application in iconified form.
	`-name`	name	Specify a name for the application being run.
`-rv`	`-reverse`	reverseVideo	Reverse foreground and background colors.
`+rv`		reverseVideo	Don't reverse foreground and background.
	`-selection-Timeout`	selectionTim-eout	Time-out in milliseconds within which two communicating applications must respond to one another for a selection request.
	`-synchronous`	synchronous	Enable synchronous debug mode.
	`+synchronous`	synchronous	Disable synchronous debug mode.
	`-title`	title	Specify a window title (e.g., to be displayed in a titlebar).
	`-xnllanguage`	xnlLanguage	The language, territory, and code set for National Language Support; this information helps resolve resource and other filenames.
`-xrm`		value of next argument	Next argument is a quoted string containing a resource manager specification, as described in Chapter 12, *Setting Resources*.

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

Here are the XView options. Most are preceded by a minus sign, but a few may be preceded by a plus sign to have the opposite effect.

Table 11-2. XView Generic Options

Flag	Long Flag	Argument Type	Description
-WH	-help		Print this table.
-Ww	-width	columns	Window width in pixels
-Wh	-height	lines	Window height in pixels
-Ws	-size	x y	(combination of above)
-Wp	-position	x y	Position of top left window corner
	-geometry	X geometry	X-style size, location
-WP	-icon_position	x y	Icon location
-Wl	-label	string	Window Label
	-title	string	Same as -label
	-name	string	Set application instance name to string
-Wi	-iconic	Start up iconified	
+Wi	+iconic	Start up open	
-Wt	-font or -fn	font name	Normal font name
-Wx	-scale	small | medium | large | extra_large	Relative size
-Wf	-foreground_color	red green blue	0-255, 0=no color; 255=full color
-fg	-foreground	color name	X Color specification
-Wb	-background_color	red green blue	same as -Wf
-bg	-background	color name	same as -fg
-rv	-reverse		Reverse -fg and -bg colors
+rv	+reverse		Don't reverse colors
-WI	-icon_image	filename	Take icon image from file

Table 11-2. XView Generic Options

Flag	Long Flag	Argument Type	Description
-WL	-icon_label	string	Label when iconified
-WT	-icon_font	font name	Name of font for -WL
-Wr	-display	server_name:screen	X display name
	-visual	StaticGray | Gray-Scale | StaticColor | PseudoColor| TrueColor | DirectColor	"Visual" to use
-depth	depth	1-24	Depth of visual to use
-Wdr	-disable_retained		Turn off "retained" hint
-Wdxio	-disable_xio_error_handler		Turn off XLib I/O handling
-Wfsdb	-fullscreen-debug		For debugging; avoid screengrabs
-Wfsdbs	-fullscreende-bugserver		Avoid server grabs only
-Wfsdbp	-fullscreende-bugptr		Avoid pointer (mouse) grabs only
-Wfsdbk	-fullscreende-bugkbd		Avoid keyboard grabs only
-WS	-defeateventse-curity		Turn off XLib event security (don't!)
-sync	-synchronous		Force a synchronous XLib connection
+sync	+synchronous		Make an asynchronous XLib connection
-Wd	-default	resource value	Set the X resource to value
	-xrm	resource:value	Set the X resource to value
	-lc_basiclocale	locale	Set basic locale to locale

Table 11-2. XView Generic Options

Flag	Long Flag	Argument Type	Description
	`-lc_displaylang`	locale	Set display language to locale
	`-lc_inputlang`	locale	Set input language to locale
	`-lc_numeric`	locale	Set numeric format to locale
	`-lc_timeformat`	locale	Set time format to locale

One very useful option is `-help`, which prints the list of options that a given program will accept, similar to the above table.†

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

An option that lets you specify any X Resource for which there is no command line argument is `-xrm`, which takes an argument of the form *resource:value* (XView clients also accept -Wd or -default, which takes two arguments, a resource and a value). The two following forms are equivalent:

```
cmdtool -xrm "OpenWindows.ScrollbarPlacement:right"
cmdtool -Wd "OpenWindows.ScrollbarPlacement:right"
```

In either case, the quotes are recommended to prevent the shell from trying to interpret any asterisk ("*") or other special characters.

11.1 Window Title and Application Name

The name of the program (as known to the server) and the title of the window can be specified on the command line. The `-title` option allows you to specify a text string as the title of the application's window. If your application has a titlebar or if the window manager you are using puts titlebars on windows, this string will appear in the titlebar. Window titles can be useful in distinguishing multiple instances of the same application.

The `-name` option actually changes the name by which the server identifies the program. Changing the name of the application itself (with the `-name` option) affects the way the application interprets resource files. This option is discussed further in Chapter 12, *Setting Resources*. If a name string is defined for an application, that string will appear as the application name in its icon.

† Indeed, my one complaint about the standard OpenWindows clients is that they only print the generic help; many of those that accept additional command line arguments don't display them when **-help** is given.

If you display information about currently running windows using the *xwininfo* or *xlswins* client, title strings will appear in parentheses after the associated window ID numbers. (If there is no title string but there is a name string, the name string will be displayed.)

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

11.2 Starting a Client Window as an Icon

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

```
% xterm -iconic &
```

XView clients accept the alternate form `-Wi`, for example,

```
% cmdtool -Wi &
```

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

All toolkits let you specify the location of the main window on the command line, for example using the `-geometry` command line argument. XView clients also allow you to specify the size and position of the icon on the command line. Unfortunately, the X Toolkit does not allow this. For most such clients, the size and position of the icon can be set using resource variables in an *.Xdefaults* or other resource file. (Setting the icon geometry in a resource file is highly recommended if you are starting the login *xterm* window as an icon.) See the appropriate client reference pages in Part Three of this guide for a complete list of available resources. Refer to Chapter 12, *Setting Resources*, for instructions on how to set resources.

11.3 Specifying Fonts on the Command Line

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

```
% cmdtool -fn fixed &
```

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

Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, describes the available screen fonts and font naming conventions. See also *xlsfonts*.

11.4 Reverse Video

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

The `+rv` option is used to override any reverse video request that might be specified in a resource file. (See Chapter 12, *Setting Resources.*) This is important, because not all clients handle reverse video correctly, and even those that do usually do so only on black and white displays.

11.5 Specifying Color

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

–`bg` ***`color`*** Sets the background color.

–`fg` ***`color`*** Sets the foreground color.

–`bd` ***`color`*** Sets the border color.

By default, the background of an application window is usually white and the foreground black, even on color workstations. You can specify a new color using either the color names listed in a system file called *rgb.txt* (described later) or hexadecimal values representing colors. The hexadecimal format is described in the section *Hexadecimal Color Specification* later in this chapter.

Many X Toolkit clients accept a `-bd` option that is intended to specify the color of the window border. However, under OPEN LOOK using the OPEN LOOK Window Manager, this customization is generally useless: the *olwm* frame effectively replaces most window borders. As an alternative, you can change the color of all windows' frames by specifying resources for *olwm* in a *.Xdefaults* or *.Xresources* file in your home directory. For more information, see Chapter 13, *Customizing olwm*, and the *olwm* reference page in Part Three of this guide.

In the next section, we'll take a look at some of the color names you can use. For now, let's consider the syntax of a command line specifying an *xterm* to be displayed in three colors:

```
% xterm -bg lightblue -fg darkslategrey -bd plum &
```

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

At the command line, a color name should be typed as a single word (for example, `darkslategray`). However, you can type the words that make up a color name separately if you enclose them in quotes, as in the command line:

```
% xterm -bg "light blue" -fg "dark slate grey" -bd plum &
```

As we'll see, the *rgb.txt* file contains variants of the same color name (for example, "navy blue" and "navyblue," or "grey" and "gray") to allow a range of spelling, spacing, and capitalization on the command line.

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

To see how a given color looks "in the large", you can put it onto your root window using the *xsetroot* client described in Chapter 14, *Customization Clients*. For example,

```
% xsetroot -solid "light sea green"
```

11.5.1 The rgb.txt File

The *rgb.txt* file, usually located in */usr/lib/X11*, is supplied with the standard distribution of X and consists of predefined colors assigned to specific text names.

A corresponding compiled database lives in two files called *rgb.dir* and *rgb.pag*. This database contains the definitions used by the server; this machine-readable file serves as a color name database and is discussed more fully in The *rgb.txt* file is the human-readable equivalent.

11.5.2 X11 Release 4 Color Names

The default *rgb.txt* file shipped with Release 4 of X contains 738 color name definitions. This number is slightly deceptive, since a number of the color names are merely variants of another color name (differing only in spelling, spacing, and capitalization).

Still, the number of colors available in Release 4 is more than double the number available in Release 3. Some of the Release 4 colors are entirely new (such as snow and misty rose) but many are just slightly different shades of colors available in prior releases.

For example, the Release 3 *rgb.txt* file includes the color sea green. The Release 4 *rgb.txt* file offers several shades of that color:

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

Each of these names corresponds to a color definition. (This list does not include the variants SeaGreen, LightSeaGreen, MediumSeaGreen, and DarkSeaGreen, which also appear in the file.) As you can see, some of these shades are distinguished in the fairly traditional

way of being called "light," "medium," and "dark." The light, medium, and dark shades of a color can probably be distinguished from one another on virtually any monitor.

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

By supplying many different shades of a single, already fairly precise color like sea green, X developers have tried to provide definitions that work well on a variety of commonly used monitors.†

You may have to experiment to determine which colors (or shades) display best on your monitor.

The color names in the Release 4 *rgb.txt* file are too numerous to list here. Although there are no literal dividers within the file, it can roughly be considered to fall into three sections:

Section 1: A standard spectrum of colors, many available in or similar to colors in Release 3 (such as sea green). These colors seem to be ordered roughly as: off-whites and other pale colors, greys, blues, greens, yellows, browns, oranges, pinks, reds, and purples.

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

Section 3: One hundred and one additional shades of grey, numbered 0 through 100 (also available in Release 3). This large number of precisely graduated grays provides a wide variety of shading for monochrome screens.

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

Section 1:
```
ghost white peach puff lavender blush lemon chiffon slate grey midnight
blue cornflower blue medium slate blue dodger blue powder blue turquoise
pale green lawn green chartreuse olive drab lime green khaki light
yellow goldenrod indian red sienna sandy brown salmon coral tomato hot
pink maroon violet red magenta medium orchid blue violet purple
```
Section 2:
```
snow1 - 4 bisque1 - 4 cornsilk1 - 4 honeydew1 -4 azure1 - 4 SteelBlue1
- 4 DeepSkyBlue1 - 4 LightCyan1 - 4 PaleTurquoise1 - 4 aquamarine1 - 4
PaleGreen1 - 4 DarkOliveGreen1 - 4 SpringGreen1 -4 gold1 - 4 RosyBrown1
- 4 burlywood1 - 4 chocolate1 - 4 firebrick1 - 4 DarkOrange1 - 4
OrangeRed1 - 4 DeepPink1 - 4 PaleVioletRed1 - 4 plum1 - 4 DarkOrchid1 - 4
```

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

```
Section 3:
grey0 (gray0)  through grey100 (gray100)
```

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

```
% showrgb | more
```

See Volume Eight, *X Window System Administrator's Guide*, for information on customizing color name definitions.

Keep in mind that colors look different on different monitors. The *xcol* and *xcoloredit* clients, from the user-contributed distribution, allow you to display the colors defined in the *rgb.txt* file. *xcol* can also edit the color specifications in a resource file. See the *xcol* client reference page in Part Three of this guide.

11.5.3 Alternative MIT X11 Release 4 and 5 Color Databases

In addition to the standard color database described previously, Release 4 also includes three other databases your system administrator can compile. These files can be found in the general release in the directory *./rgb/others*.

raveling.txt Designed by Paul Raveling, this database rivals the default database in size and scope but was tuned to display optimally on Hewlett-Packard monitors.

thomas.txt Based on the Release 3 database, this file has been modified by John Thomas of Tektronix to approximate the colors in a box of Crayola Crayons.

old-rgb.txt This is nothing more than the Release 3 database.

11.5.4 MIT X11R5 Color Extensions

The MIT X11 Release 5 system contains numerous extensions for color editing and specification. These are not described here yet.. If your system supports them, you will find a full discussion in Volume Eight, *X Window System Administrator's Guide*.

11.5.5 Hexadecimal Color Specification

You can specify colors more exactly than is possible with the names in the *rgb.txt* file by using a hexadecimal color string. You probably won't use this method unless you require a color not available by using a color name. In order to understand how this works, you may need a little background on how color is implemented on most workstations.

11.5.5.1 The RGB Color Model

Most color displays on the market today are based on the RGB color model. Each pixel on the screen is actually made up of three phosphors: one red, one green, and one blue. Each of these three phosphors is excited by a separate electron beam. When all three phosphors are fully illuminated, the pixel appears white to the human eye. When all three are dark, the pixel appears black. When the illumination of each primary color varies, the three phos-

phors generate a subtractive color. For example, equal portions of red and green, with no admixture of blue, makes yellow.

As you might guess, the intensity of each primary color is controlled by a three-part digital value—and it is the exact makeup of this value that the hexadecimal specification allows you to set.

Depending on the underlying hardware, different servers may use a larger or smaller number of bits (from 4 to 16) to describe the intensity of each primary. To insulate you from this variation, most clients are designed to take color values containing anywhere from 4 to 16 bits (1 to 4 hex digits), and the server then scales them to the hardware. As a result, you can specify hexadecimal values in any one of these formats:

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

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

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

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

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

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

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

The bottom line here is that if you don't intimately know the physics of color, the best you can do is look up existing colors from the color name database and experiment with them by varying one or more of the primaries till you find a color you like. Unless you need precise colors, you are probably better off using color names.

11.5.5.2 How Many Colors are Available?

The number of distinct colors available on the screen at any one time depends on the amount of memory available for color specification. (The *xdpyinfo* client provides information about a display, including the number of colors available at one time. See Chapter 8, *Other Standard Clients*, and the *xdpyinfo* reference page in Part Three of this guide for details. As well, the Sun OpenWindows server lets you control the number of colors with its **-cubesize** command line argument; see the *OpenWindows Server Guide* for details.)

A color display uses multiple bits per pixel (also referred to as multiple planes or the *depth* of the display) to select colors. Programs that draw in color use the value of these bits as a pointer to a lookup table called a *colormap*, in which each entry (or *colorcell*) contains the RGB values for a particular color.* As shown in Figure 11-1, any given pixel value is used as an index into this table—for example, a pixel value of 16 will select the 16th colorcell.

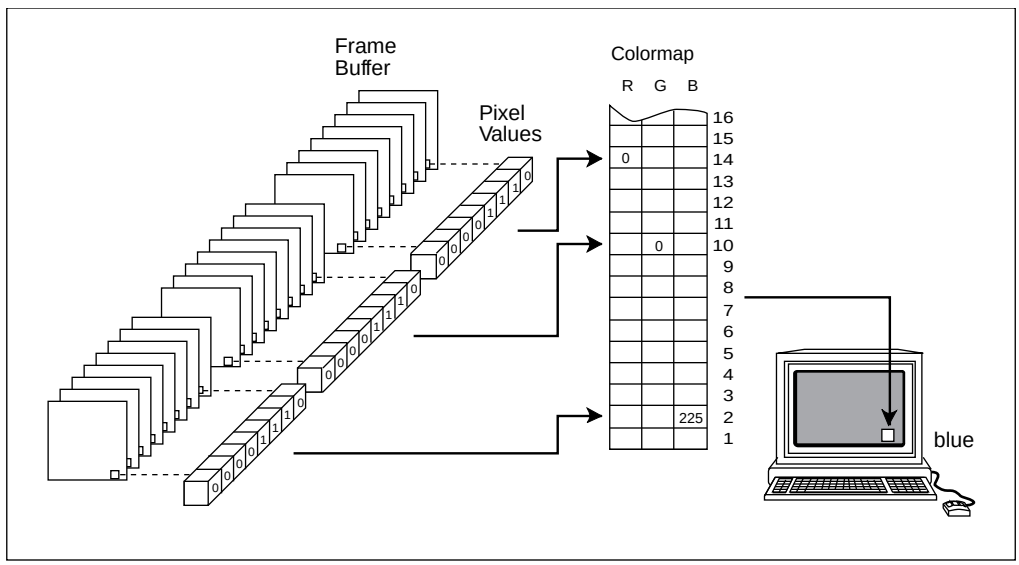


Figure 11-1. Multiple planes used to index a colormap

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

First, the range of colors possible on the display is a function of the number of bits available in the colormap for RGB specification. If 8 bits are available for each primary, then

the range of possible colors is 256^3 (more than 16 million colors). This means that you can create incredibly precise differences between colors.

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

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

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

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

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

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

To top it off, there are even clients that may temporarily swap in a private colormap of their own. If this happens, all other applications will be displayed in unexpected colors because of the way color is implemented. This is called *colormap flashing*, because the colormaps flash in and out as the pointer enters and leaves such clients. You can "lock" in the colormap of a client under the OpenWindows version of *olwm* using the Colormap Lock key (Ctrl/L2 on the Sun keyboard).

In order to minimize such conflicts, you should request precise colors only when necessary. By preference, use color names or hexadecimal specifications that you specified for other applications.

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

11.6 Border Width

Many clients accept a `-bw` option that is intended to specify the width of the window border in pixels. However, as in the case of the `-bd` (border color) option, under OPEN LOOK, this customization is generally useless because the *olwm* window frame effectively replaces most window borders.

As an alternative, you *can* change the width of the frame of all windows by specifying resources for *olwm* in a *.Xresources* or *.Xdefaults* file in your home directory. For more information, see Chapter 13, *Customizing olwm*, and the resources on the *olwm* reference page in Part Three of this guide.

11.7 xterm and cmdtool example

Here is an example that consolidates several of the options that we've discussed so far. We want a terminal emulator, with a window size of 80 columns by 48 rows (double the size of a conventional "24x80" ASCII terminal), at the left margin but with its top 100 pixels down from the top of the screen. And we want the name in its titlebar to be whatever is stored in the shell variable "HOSTNAME". If we prefer to use *xterm*, we could use this form:

```
xterm -geometry 80x48+0+100 -T "$HOSTNAME"&
```

If we are using *cmdtool* we could give this specification:

```
cmdtool -Ww 80 -Wh 40 -Wp 0 100 -Wl "$HOSTNAME"&
```

However, this attempt at an X Toolkit-compatible one:

```
cmdtool -geometry 80x48+0+100 -title "$HOSTNAME"&
```

will fail, because in this context the width and height are interpreted in pixels, so we get a very tiny window, about an inch high! By contrast, *xterm* interprets the geometry size in columns and rows, while *cmdtool* interprets it in pixels, and provides the alternate *-width* and *-height* command line arguments (columns and rows, respectively), so that you can specify it either way.

11.8 Summary

We've now shown most of the standard command line options for the Intrinsics toolkits (Athena and OLIT) as well as for the XView toolkit. Since color is so important in making a display visually appealing, we spent some time on it. In the next chapter we describe Resources, which are a way you can control the behavior of window clients without having to type your preferences each time you start that client.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 12

Setting Resources

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

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

Traditional UNIX applications rely on command line options to allow users to customize the way they work. As we've already discussed in Chapter 11, *Command Line Options*, X applications support command line options too, but often not for all features. Also, there can be so many customizable features in an application that entering a command line to set them all would be completely impractical. (Imagine the aggravation of misspelling an option in a command that was three lines long!)

X offers an alternative to customizing an application on the command line. Almost every feature of a program can be controlled by a variable called a *resource*; you can change the behavior or appearance of a program by changing the *value* associated with a resource variable. (All of the standard X Toolkit *Command Line Options* described in Chapter 11 have corresponding resource variable names. See Table 11-1 and Table 11-2 for more information.)

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

individual *instance* of an object. This is illustrated by these resource specifications for a hypothetical application called *xclient*:

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

The first resource specification makes the foreground color of all buttons in the *xclient* application (in the class `Buttons`) blue; the second resource specification makes the foreground color of the `help` button in this application (an instance of the class `Buttons`) red.

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

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

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

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

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

12.1 Resource Naming Syntax

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

Most of the common clients are written to use the X Toolkit. As described in Chapter 8, *Other Standard Clients*, toolkits are a mechanism for simplifying the design and coding of applications, and making them operate in a consistent way. Toolkits provide a standard set of objects, or widgets, such as menus, command buttons, dialog boxes, scrollbars, and so on. As we'll see, the naming syntax for certain resources parallels the object hierarchy that is built into X Toolkit programs.†

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

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

```
xterm*scrollBar: True
```

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

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

12.1.1 Syntax of Toolkit Client Resources

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

object.subobject[.subobject... .attribute : value

where:

† If a client was built with the X Toolkit, this should be noted on the reference page. In addition to certain application-specific resource variables, most clients that use the X Toolkit recognize a common set of resource variables, listed in Table 12-1.

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

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

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

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

The type of ***value*** to supply is often evident from the name of the resource or from the description of the resource variable on the reference page. Most of these values are similar to those used with the command line options described in Chapter 9.

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

12.1.2 Tight Bindings and Loose Bindings

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

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

A tight binding means that the components on either side of the dot must be next to one another in the widget hierarchy. A loose binding is signaled by an asterisk, a wildcard character which means there can be any number of levels in the hierarchy between the two surrounding components.

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

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

```
xterm.scrollBar:True
```

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

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

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

```
xterm*scrollBar:True
```

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

12.1.3 Instances and Classes

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

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

or:

```
xterm*Foreground:darkblue
```

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

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

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

where `Buttons` is a class name and the `delete` button is an instance of the `Buttons` class. This type of specification works because an instance name always overrides the corresponding class name for that instance. Class names thus allow default values to be specified for all instances of a given type of object. Instance names can be used to specify exceptions to the rules outlined by the class names. Note that a class name can be used with a loose binding to specify a resource for all clients. For example, this specification would say that the foreground colors for all clients should be blue:

```
*Foreground:blue
```

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

12.1.4 Precedence Rules for Resource Specification

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

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

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

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

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

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

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

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

1. The levels in the hierarchy specified by the user must match the program's expectations or the entry will be ignored. For example, if the program expects either:
   xterm.vt100.scrollBar:***`valueinstance name`*** `or:`
   XTerm.VT100.ScrollBar:***valueclass name***

   then the resource specification:
   `xterm.scrollBar: True`
   won't work, because the tight binding is incorrect. The objects `xterm` and `scrollBar` are not adjacent in the widget hierarchy: there is another widget, `vt100`, between them. The specification would work if you used a loose binding, however:
   `xterm*scrollBar: True`
   `(Note that the class name of` *`xterm`* `is XTerm`, not `Xterm` as you might expect.)

2. Tight bindings take precedence over loose bindings. That is, entries with instance or class names prefixed by a dot are more specific than entries with names prefixed by an asterisk, and more specific entries take precedence. For example,

the entry `xterm.vt100.geometry` will take precedence over the entry `xterm*geometry`.

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

4. An instance or class name that is explicitly stated takes precedence over one that is omitted. For example, the entry `xterm*scrollbar` is more specific than the entry `*scrollBar`.

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

To illustrate these rules, let's consider the following resource specifications for the hypothetical Toolkit application *xclient*, shown in Example 12-1.

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

Example 12-1. Sample resources

The program would try to match these specifications against these complete tightly bound instance and class specifications:

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

Note that these specifications are the instance and class names for the same resource. Each component of the instance name belongs to the class in the corresponding component of the class name. Thus, the instance `toc` occurs in the class `Box`, the `messageFunctions` instance name is from the class `SubBox`, etc.

Both resource specifications in Example 12-1 match these instance and class names. However, with its tight bindings and instance names,
`xclient.toc*Command.active%Foreground`
matches more explicitly (i.e., with higher precedence). The resource is set: the `foreground` color of the `include` button in its `active` state is set to `black`.

The specification `*Command.Foreground` also matches the instance and class names but is composed entirely of class names which are less specific; thus, it takes lower precedence than the first line in Example 12-1 (which sets the `include` button to `black`).

However, since the second line is an acceptable specification, hypothetically it would set the foreground color of other objects in the `Command` class. This resource would be set for *xclient*, as well as any other application, since the line begins with the asterisk wildcard. So if there were other *xclient* command buttons comparable to the `include` button in the hierarchy, this second line would set the foreground color of these buttons to `green`. If you want a more detailed description of how resource precedence works, see Section 9.2.3 of Volume Four, *X Toolkit Intrinsics Programming Manual*.

12.1.5 Some Common Resources

Each Toolkit command line option (listed in Table 11-1, ". MIT Toolkit and OLIT Standard Options," on page 283) has a corresponding resource variable. Most X Toolkit (and OLIT Toolkit) applications recognize some subset of these resources.

Table 12-1 lists the resource variables recognized by most Toolkit clients.

Table 12-1. Common Toolkit Resources

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

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

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

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

Of course, the specification:

```
xclient*background
```

would match any and all of them.

Appendices list resources for the OLIT and Athena widgets.

12.2 Event Translations

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

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

For many clients, the resource manager recognizes mappings between certain input events (such as a pointer button click) and some sort of action by the client program (such as

selecting text). A mapping between one or more events and an action is called a *translation*. A resource containing a list of translations is called a *translation table*.

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

12.2.1 The Syntax of Event Translations

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

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

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

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

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

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

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

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

```
<event>: action
```

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

These are default translations for *xterm*.‡

† For more information on events and translations, see Volume Four, *X Toolkit Intrinsics Programming Manual.*

‡ They are actually slightly simplified versions of default translations. Before you can understand the actual translations listed on the *xterm* reference page in Part Three of this guide, you must learn more about the syntax of translations. In addition to the current chapter, read Appendix F, *X Toolkit Translation Table Syntax*.

All of the events are simple, comprised of a single button motion. As we'll see, events can also have modifiers: i.e., additional button motions or keystrokes (often Control or Meta) that must be performed with the primary event to produce the action. (Events can also have modifiers that *must not* accompany the primary event if the action is to take place.)

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

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

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

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

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

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

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

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

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

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

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

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

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

12.2.1.1 xterm Translations to Use xclipboard

As explained in Chapter 5, the *xclipboard* client provides a window in which you can store text selected from other windows. You can also paste text from the *xclipboard* window into other windows. See the discussion of *xclipboard* in Chapter 8, *Other Standard Clients* before proceeding .

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

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

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

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

The second line modifies the way selected text is pasted into a window so that the `CLIPBOARD` selection can be pasted. As described in Chapter 5, pressing the second pointer button pasted the contents of the PRIMARY selection, by default. If there is no PRIMARY selection, the contents of the cut buffer are pasted. The default translation that sets this behavior is the following:

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

This translation specifies that releasing pointer button 2, while pressing any modifier button or key other than Control or Meta, inserts text from the PRIMARY selection or, if the selection is empty, from cut buffer 0. In the second line of our translation table, we've replaced `CUT_BUFFER0` with the `CLIPBOARD` selection. The new behavior is that releasing the second pointer button pastes the PRIMARY selection, or if there is none, the CLIPBOARD selection.

Thus, according to the translations in the example, if you select text as usual with the first pointer button, and then additionally press the third button (while continuing to hold down the first button), the text becomes the CLIPBOARD selection and appears automatically in the *xclipboard* window, as shown in Figure 12-1.

Figure 12-1. Selected text appears automatically in the xclipboard window

Since our first translation specifies a different event/action mapping than the default translation for selecting text (discussed in the previous section), the default translation still applies. If you select text with the first pointer button alone, that text is still made the PRI-

MARY selection and fills CUT_BUFFER0. To send text to the *xclipboard*, you would need to press the third pointer button as well; thus, not all selected text needs to be made the CLIPBOARD selection (and sent automatically to the *xclipboard*).

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

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

12.2.1.2 Entering Frequently Used Commands with Function Keys

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

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

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

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

The translation table would be:

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

This sample translation causes `lpq -Pprinter1` to be passed to the command line in the active *xterm* window when you press the F1 function key, as in Figure 12-2.

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

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

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

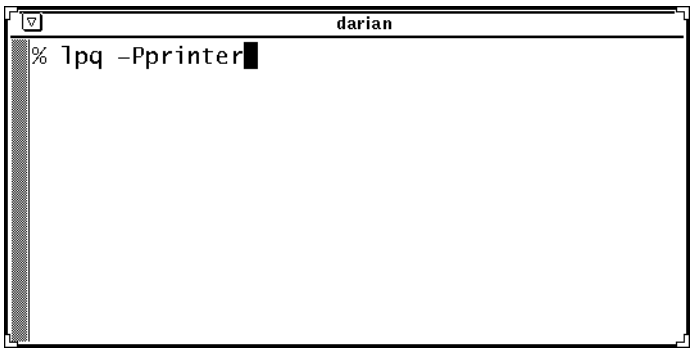


Figure 12-2. Pushing F1 passes command text to xterm shell .

```
0xffab
0x00ab
```

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

The following translation table specifies that pressing F1 passes the line `lpq -Pprinter1` followed by a carriage return to an *xterm* window:

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

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

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

According to these translations, pressing F2 inserts the command string

```
cd ~/bitmap;ls
```

which changes directory to *~/bitmap* and then lists the contents of that directory. Notice that you can issue multiple commands (*cd* , *ls*) with a single key. Pressing F3 changes directory to */usr/lib/X11*.

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

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

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

12.2.1.3 Other Clients that Recognize Translations

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

You can also modify the operation of the Text widget used by *xedit*, *xmh*, and other X Toolkit applications. See Appendix I, *Athena Widget Resources*, for a list of actions recognized by the Text widget. Keep in mind, however, that the default Text widget recognizes dozens of commands, which are summarized in the discussion of *xedit* in Chapter 8, *Other Standard Clients*. It may not be practical or desirable to modify them all.

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

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

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

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

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

For example, the following table offers two suitable translations:

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

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

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

12.3 How to Set Resources

Learning to write resource specifications is a fairly manageable task, once you understand the basic rules of syntax and precedence. In contrast, the multiple ways you can set

resources—for a single system, for multiple systems, for a single user, for all users—can be confusing. For our purposes, we are primarily concerned with specifying resources for a single user running applications both on the local system and on remote systems in a network.

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

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

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

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

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

12.3.1 A Sample Resources File

Figure 12-3 shows a sample resources file. This file sets the border width for all clients to a default value of two pixels, and sets other specific variables for *xclock* and *xterm*. The meaning of each variable is obvious from its name (for example, `xterm*scrollBar: True` means that *xterm* windows should be created with a scrollbar.

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

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

```
*borderWidth: 2
!
! xclock resources
!
xclock*borderWidth: 5
xclock*geometry: 64x64
!
```

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

Figure 12-3. A sample resources file

12.3.2 Specifying Resources from the Command Line

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

12.3.2.1 The –xrm Option

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

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

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

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

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

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

```
xterm*pointerShape: pirate
```

This command line specification of another cursor will fail:

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

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

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

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

12.3.2.2 The –name Option

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

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

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

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

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

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

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

```
% xterm &
```

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

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

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

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

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

12.4 Setting Resources with xrdb

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

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

`xrdb` [***`options`***] [***`filename`***]

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

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

The resource ***filename*** can be anything you want. Two commonly used names are *.Xdefaults* and *.Xresources*. The former is an older form, but is more commonly used in the OpenWindows environment. The latter is newer, and is more prevalent in X11R5 environments. Either form can be used.

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

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

12.4.1 Querying the Resource Database

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

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

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

12.4.2 Loading New Values into the Resource Database

By default, *xrdb* reads its input (either a file or standard input) and stores the results into the resource database, replacing the previous values. If you simply want to merge new values with the currently active ones (perhaps by specifying a single value from standard input), you can use the `-merge` option. Only the new values will be changed; variables that were already set will be preserved rather than overwritten with empty values. In fact, if you are running with *xdm*, it is preferable to use -merge, since this will prevent you from accidentally erasing some valuable initial resources that *xdm* might have stored in the server.

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

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

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

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

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

```
xterm*scrollBar:True
```

and the more specific value:

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

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

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

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

12.4.3 Saving Active Resource Definitions in a File

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

You don't need to edit the file manually (although you certainly could.) The `-edit` option allows you to write the current value of the RESOURCE_MANAGER property to a file. If the file already exists, it is overwritten with the new values. However, *xrdb* is smart enough to preserve any comments and preprocessor declarations in the file being overwritten, replacing only the resource definitions.

For example:

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

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

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

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

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

12.4.4 Removing Resource Definitions

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

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

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

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

```
% vi filename
```

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

```
% xrdb -load filename
```

12.4.5 Listing the Current Resources for a Client: appres

The *appres* (*app*lication *res*ource) program, available as of X11 Release 4, lists the resources that currently might apply to a client. These resources may be derived from several sources, including the user's *.Xresources* file and a system-wide application defaults file. The directory */usr/lib/X11/app-defaults* contains application default files for several clients. The function of these files is discussed in the next section. For now, be aware that all of the resources contained in these files begin with the class name of the application.

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

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

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

```
% appres XTerm
```

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

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

```
% appres XTerm bigxterm
```

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

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

```
% appres xterm
```

might return resources settings similar to these:

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

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

12.4.6 Other Sources of Resource Definition

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

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

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

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

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

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

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

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

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

   These methods are used to set machine-specific resources.

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

All of these various sources of defaults will be loaded and merged according to the precedence rules described earlier in the section "Precedence Rules for Resource Specification."

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

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

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 13

Customizing the OPEN LOOK Window Manager

Some window managers have the flexibility to let you paint your root window sky blue by pressing the middle pointer button in a titlebar while holding down the shift, control, and ALT keys. As we've seen in Chapter 2, *Working in the OPEN LOOK Environment*, the OPEN LOOK User Interface specifies the window system's behavior in such detail that the user can't be given this degree of flexibility.Hence, The OPEN LOOK Window Manager does not have the blazing configurability of some other X11 Window Managers. It is designed to allow users flexibility, but still keep you within the confines of the OPEN LOOK Graphical User Interface specification.

If you had that kind of flexibility, users could (and some would) configure their systems in ways that would violate the OPEN LOOK specification. So OPEN LOOK Window Manager is considerably easier to configure—or less flexible, depending on your point of view—than some other X Window System window managers, such as the widely-used *twm* and *mwm*. Customization of *olwm* is, in fact, limited to changing or replacing the *root menu* or *Workspace Menu*, specifying options on the command line, and specifying a number of X Resources that control *olwm*'s behavior. These topics will occupy our attention in the remainder of this chapter.

13.1 The Workspace Menu

The OPEN LOOK specification describes two levels of conformance, called *Level One* and *Level Two*. Level One describes a basic, minimal implementation, while Level Two has more features than Level One. The AT&T-OL version of *olwm* conforms to Level One, which only requires that you be able to change the list of programs in the *Programs* part of the *Workspace Menu*. The OpenWindows version of *olwm* conforms to Level Two, which specifies that the user can replace *all* the entries in the *Workspace Menu*. The following section describes both versions.

13.1.1 Level 1 Customization (AT&T-OL)

The list of Programs in the *Programs* sub-menu of the AT&T-OL *Workspace* menu is maintained from the Properties program. Selecting the Category *Programs Submenu* produces a window that lets you add additional programs. There is a Scrolling List listing the optional programs, a *Name* and *Invocation String* and *Mnemonic* text field that lets you put in the values for each program. Below this are several buttons.

To add a new program, for example, select Insert. Type the name (such as "Local Xterm") in the *Name* text field. Type the invocation string (such as "exec xterm") in the *Invocation String* text field. Pick a single-character name that isn't already a mnemonic for one of the programs, and type it in the *Mnemonic* field. Then click SELECT on *Apply Edits* and your changes will be saved. From now on, the *Programs* submenu of the *Workspace* menu will include this item.

You can use the scrolling list in the normal way to edit or delete items in the scrolling list. To change and item, for example, click SELECT on it in the scrolling list. Update one or more of the text fields (remember to click on each field to activate it), and click SELECT on the *Apply* button to save your changes. To delete an item, select it, click on the DELETE button, and click SELECT on the *Apply* button to make your changes take effect.

13.1.2 Level 2 Customization (OpenWindows)

To customize the *Workspace* menu in OpenWindows, you need only edit a text file. The default file is */usr/openwin/lib/openwin-menu*, which looks like this:

```
# # @(#)openwin-menu23.15 91/09/14 openwin-menu # #OpenWindows
default root menu file - top level menu #
"Workspace" TITLE
"Programs" MENU$OPENWINHOME/lib/openwin-menu-programs
"Utilities" MENU$OPENWINHOME/lib/openwin-menu-utilities
"Properties..."PROPERTIES
SEPARATOR
"Help..."exec $OPENWINHOME/bin/helpopen handbooks/top.toc.handbook
"Desktop Intro..."exec $OPENWINHOME/bin/helpopen
handbooks/desktop.intro.handbook
SEPARATOR
"Exit..."EXIT
```

This is what *olwm* actually uses to create the normal workspace menu, consisting of:

Table 13-1. Olwm normal root window

Programs
Utilities
Properties... -
-
Help Desktop Intro...

Table 13-1. Olwm normal root window

-
Exit...

Each line in the menu text file consists of three fields:† the name to appear on the menu, optional keywords, and the action to take when the line is selected. Comment lines beginning with '#' in column one, and null lines, are ignored (you can usefully put a '#' before a line that is temporarily not working, to make it invisible). Look at the example a minute and the overall pattern should become clear.

The easiest and safest way to change this file is to copy it into your home directory under the name *.openwin-menu*, for example, */home/darian/ian/.openwin-menu*. Then make any changes you want. But make changes one at a time, and keep track of your changes (hint: use RCS or SCCS). Why? Because if you make a mistake, and your menu is incorrect, *olwm* may just ignore the menu file completely, and use the system one (which is why you should not tamper with the system copy). If your menu file is huge, it can be hard to spot errors. The current *olwm* is pretty good at reporting many errors:

```
olvwm: menu label mismatch in file /home/xyz/ian/.openwin-menu, line 104
```

But it's still easier to make changes in small doses so you can easily back up and see what you did wrong.

As an example of a simple change, if you'd prefer to start *xterm* instead of *cmdtool* from the *Programs* menu, change the line

```
"Command Tool..." DEFAULTexec $OPENWINHOME/bin/xview/cmdtool
```

to these two lines:

```
"Xterm..." DEFAULTexec $OPENWINHOME/demo/xterm
"Command Tool..." exec $OPENWINHOME/bin/xview/cmdtool
```

The DEFAULT keyword is optional, but useful: it specifies what menu item will be selected when you click or release SELECT on the parent menu, in this case, on the word *Programs* in the top-level *Workspace* menu. The word "exec" is present as an optimization. A copy of the UNIX shell is used to interpret the string that is given as the command, and the *exec* prevents this extra shell from sticking around waiting for the application *xterm*, for example) to terminate.

As another example, If you get tired of the "Please confirm exit from window system" prompt, you can suppress it by changing

```
"Exit" EXIT
```

to

```
"Exit!" EXIT_NO_CONFIRM
```

† The syntax is the same as the older SunView root menu, except that you cannot (yet?) specify an icon in place of the name field.

You probably should have both; the latter will exit without confirming. Use with care!

There are several keywords that can be specified, listed in Table 13-2. They must be in upper case.

Table 13-2. Sun OPEN LOOK Window Manager menu file keywords

Keyword	Function
TITLE	Specify title for menu (or submenu).
MENU, END	Make a menu, or invoke one as in examples above
DEFAULT	Specify default case for menu
PIN	(after END) makes menu pinnable.
REFRESH	Refresh all windows
POSTSCRIPT	Send rest of line to NeWS (OpenWindows only)
SAVE_WORKSPACE	Save windows in $HOME/.openwin-init
PROPERTIES	Start up Workspace or Properties manager
EXIT	Exit window system
EXIT_NO_CONFIRM	Exits without confirming notice.
WM_EXIT	Exit Window Manager, but don't kill clients.
RESTART	Re-run olwm (e.g., after changing the menu file)
FLIPDRAG	Toggle dragging of window contents when moving windows.
FLIPFOCUS	Flip focus policy (click vs. follow-mouse)
NOP	Does nothing (placeholder).
SEPARATOR	Produce a blank line.
PRINT_SCREEN	Not implemented yet.
CLIPBOARD	Not implemented yet.
WINDOW_CONTROLS	Not implemented yet.
REREAD_MENU_FILE	Just what it says

Table 13-3

You may not like the default root window menu, so you can completely change it. My own root menu looks like this: box; l. Local Windows Window Programs OpenWin Demos

Refresh Screen Lock Screen Terminations The first three pull down menus. The Local Windows lets me at several different terminal emulators. Another menu, normally commented out but made available when I'm on a LAN or the Internet, has remote logins for the hosts I normally use,

```
"ozzify"xterm -T ozzify -geom +200+140 -e rlogin ozzify
```

for each one. The *Window Programs* menu has a **long** list of X11 (and NeWS, and SunView) applications. Some are straight invocations, like

```
"Author/Editor"exec $SQBIN/ae
```

Others are more involved, like running programs remotely with a display back to my workstation. These three entries let me use *xman* (see Chapter 8, *Other Standard Clients*) on any of four different versions of SunOS:

```
"Man pages (SunOS3.5)" rsh sq /usr/bin/X11/xman -display $DISPLAY
"Man pages (SunOS4.0)" rsh sqarc /usr/bin/X11/xman -display $DISPLAY
"Man pages (SunOS4.1)" xman -display $DISPLAY
"Man pages (SunOS5.0)" rsh sqlaris /usr/bin/X11/xman -display $DISPLAY
```

There is no limit to what you can put in this file; review the discussion of DISPLAY in Chapter 1, *An Introduction to OPEN LOOK and the X Window System* and the examples here. As a final example, to try out some of the built-in functions described earlier, and to confirm that certain functions were still unimplemented, we just added this to our own menu file:

```
TestMENU "Weird stuff"TITLE
    FlipFocusFLIPFOCUS
    FlipDragFLIPDRAG
    winctl  WINDOW_CONTROLS
    prtscr  PRINT_SCREEN
TestEND PIN
```

Remember that you have to re-start the window manager for changes to take effect; that's why our have an entry that calls RESTART on the top-level *Workspace* menu. A hint: if you make any mistakes, your entire structure of menu files will be ignored. To avoid having to restart olwm when modifying your own files, you may wish to modify the system Utilities menu *$OPENWINHOME/lib/openwin-menu-utilities* to include the following lines:

```
"Re-read menu file"REREAD_MENU_FILE
"Restart Window Manager"RESTART
```

If you are having trouble with your *.openwin-menu* file and want to see the error messages from programs run from it, you can exit your window manager and restart olwm from a visible cmdtool, as follows:

Either choose "Exit Window Manager" if you've installed WMEXIT in your Terminations or Utilities menu, or, give the command *ps -ax* (*ps -ae* on Solaris 2 or System V), and use the UNIX kill command to kill the olwm process, for example,

```
% ps -ax
 PID TT STAT TIME COMMAND
 0 ? D 0:00 swapper
 1 ? IW 0:00 /sbin/init -
 2 ? D 0:00 pagedaemon
```

```
    ...
    172 co IW 0:00 olwm
    174 co IW 0:00 olwmslave
    176 co I 0:00 clock -Wn -Wp 930 90 -digital
    178 p1 S 0:09 cmdtool -Ww 80 -Wh 24 -Wl darian -scale extra_l
    298 p1 R 0:00 ps -ax
  % kill 172
  %
```

Then give the command

```
% olwm -display :0 &
```

from within the terminal emulator. This will start a new copy of olwm, with its standard output and standard error displayed in the terminal window. Then when olwm has trouble reading or parsing your menu file, or a program you start doesn't work, you can see why.

13.2 OPEN LOOK Window Manager Command Line Options

13.2.1 OPEN LOOK Window Manager Options - AT&T-OL Version

The AT&T-OL version of *olwm* has very simple command-line customization. In fact, it accepts only one optional command-line argument: the name of the display to use. It defaults to the display named in the DISPLAY environment variable setting; if this is not set, it defaults to `unix:0`, that is, the console of the 386 PC (or other workstation) on which it is invoked.

13.2.2 OPEN LOOK Window Manager Options - OpenWindows Version

The following is a list of command-line options for the OpenWindows version of *olwm.* Most have counterparts in the X11 resource database (see Chapter 12, *Setting Resources*, and the section *Configuring OLWM* with resources below). A command-line option will override the corresponding setting from the resource database.

`-2d` Use two-dimensional look. This is the default, and the only possible look, for monochrome displays.

`-3d` Use three-dimensional look. This is the default for a color screen, and is ignored on a monochrome screen

You can usually ignore the two options above, since the window manager usually does the right thing.

`-c, –click`. Use click-to-focus mode. This is the default focus mode. The opposite of `-follow.`

`-f, –follow` Use pointer-driven (focus-follows-mouse) focus policy. Default mode is click-to-focus.

You normally set this in an X11 resource file; you can specify the other policy on the command line to try it out, for example.

`-parent` Ignore windows that are already on the screen. In earlier versions of *olwm* this was used to make the window manager to leave titlebars off certain "desk acces-

sory" windows. However, the preferable way to achieve this is to name them in the "MinimalDecor" resource, since windows ignored by `-parent` can never be moved or resized, and will never receive the input focus.

13.2.2.1 Debugging Options

There are several *Debugging Options* described in the *olwm* Reference Manual Page in Section Three of this guide that are only meant for people debugging X applications or *olwm* itself. Don't use them unless you know what you are doing.

13.2.2.2 Generic Options

The OpenWindows version recognizes the following generic options (see Chapter 11, *Command Line Options*):

Table 13-4. Olwm generic options

`-display` display	Specify which display to manage.
`-fn` fontname `-fontname` fontname	Set the font for window titles.
`-name` resource-name	Use name to find resources in X11 resource database.
`-xrm` resource-string	Put X11 resources on the command-line.

You would normally start *olwm* from your.*xinitrc* file, for example:

```
olwm -display unix:0 -fn zapfchancery-mediumitali c&
```

in your .*xinitrc* file.

13.3 Configuring OLWM with resources

As with most X11 programs, there is a significant amount of configurable behavior built into the window manager, and much of it can be configured with X Resources. Unfortunately, the two versions of *olwm* — AT&T-OL and Sun OpenWindows — have totally incompatible resource lists, so it is necessary to know which version you are using and to choose the appropriate resources.

13.3.1 Resources for Configuration — AT&T-OL

AT&T does not document the resources needed to configure their version of *olwm*, on the grounds that you are expected to use the Workspace Properties Manager to update them.

However, it has been determined that the following resources are at least known to their version of *olwm*.

Table 13-5. AT&T-OL Resources

Configuration Resources
PointerFocus
SaveSet
IconGridSize
IconGrid
WindowAttributes
WindowFrameColor
Warnings
HelpDirectory
WMIconGravity
HeaderFont

Table 13-6. AT&T-OL Resources that set Key Values

Keys for shortcuts
wmOpenCloseKey
wmSizeKey
wmPropertiesKey
wmRefreshKey
wmBackKey
wmQuitKey
wmDismissThisKey
wmDismissAllKey
wmMoveKey

Table 13-6. AT&T-OL Resources that set Key Values

Keys for shortcuts
wmResizeKey
wmOwnerKey

13.3.2 Resources for Configuration — OpenWindows

Here are the resources allowed by the Sun OpenWindows version of *olwm*. Most of these resources can be set by the Properties program (See Chapter 14, *Customization Clients*). However, they are listed here for reference, and for those who wish to edit the Resource files manually.

The "class" name for Sun's version of *olwm* is "OpenWindows", so the string "OpenWindows." must be used as part of the name in the resource file Thus, to enable automatic "raising" (move to front) of the focus window, you would put "OpenWindows.AutoRaise: true" in your X resource file.

Some resources specify key bindings. A key specification is a list of words separated by space, each of which is a KeySym name. Every word except the last must specify a modifier key. All KeySym names are case-sensitive. For example, to bind the Color-Lock key to control-shift-F1, you would use the following resource specification:

```
OpenWindows.ColorLockKey: Control Shift F1
```

You can specify that a key will have effect under all modifier combinations by using the special keyword "Any" instead of a real X11 modifier.

Boolean values can be specified in any of several ways: the pairs "true", "false", "on", "off", "yes", "no", "1", "0", "t", and "nil" will all work.

Table 13-7. OpenWindows OPEN LOOK Window Manager Resources

Resource	Type	Function
AutoRaise	boolean	Move focussed window to front
Beep	enumeration	Never, Notice or Always (==on warnings, etc.)
ButtonFont	font name	Font for buttons
ClickMoveThreshold	integer	
ColorFocusLocked	boolean	Color focus lock policy
ColorLockKey	key specification	Key to lock colormap
ColorUnlockKey	key specification	Key to unlock colormap

Table 13-7. OpenWindows OPEN LOOK Window Manager Resources

Resource	Type	Function
ConfirmKey	key specification	Key selects default from Notices
CursorFont	font name	Font for cursor (don't change)
DefaultTitle	string	Title if application has none
DragRightDistance	integer	For menus
DragWindow	boolean	Move whole window or just outline
EdgeMoveThreshold	integer	For mouse moves
FlashFrequency	integer	How often to flash after Owner?
FocusLenience	boolean	Don't enforce ICCCM on "input hint"
FrontKey	key specification	Key to front (or back) window
GlyphFont	font name	Font for drawing; don't change
IconFont	font name	Font for icon names
IconLocation	enumeration	Where to stack icons
MinimalDecor	list of strings	Desk Accessory windows; no titlebar
MoveThreshold	integer	For mouse moves
MultiClickTimeout	integer	For pointer clicks
OpenKey	key specification	Key to open (or close) a window
PPositionCompat	boolean	Compatibility for X11R3 clients
PopupJumpCursor	boolean	
RefreshRecursively	boolean	
RubberBandThickness	integer	
SaveWorkspaceTimeout	integer	
SetInput	enumeration	SnapToGrid
TextFont	font name	Font for normal text
TitleFont	font name	Font for titlebar
Use3D	boolean	Use 3D (or 2D) OPEN LOOK
Use3DFrames	boolean	Use 3D frame thickness

Table 13-7. OpenWindows OPEN LOOK Window Manager Resources

Resource	Type	Function
WindowColor	color specification	BG1 color
WorkspaceColor	color specification	Root window color, bitmap, or null

Each of these is discussed in more detail in the *olwm* reference manual page in Section Three of this book.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

CHAPTER 14

Customization Clients

This chapter describes how to personalize the appearance of your display, and the operation of your keyboard and pointer, using these clients:

Properties The OPEN LOOK Properties Editor (Workspace Manager) to update your X Resources file for such things as color settings, icon placement, menu operations, mouse settings and other features.

xset A standard X11 client to set certain characteristics of the keyboard, pointer, and display.

xsetroot A standard X11 client to set root window characteristics.

xmodmap A standard X11 client to change pointer and modifier key mappings.

xkeycapsA client to help with key mappings.

14.1 Properties Resource Editor

The OPEN LOOK properties editor is both a good example of an OPEN LOOK application and a useful tool that edits your X resources file under control of OPEN LOOK-based menus. It differs from the Motif Resource Editor *mre* described in Appendix M, *OPEN LOOK and Motif*, in that it is a fully supported part of the OPEN LOOK package, and in that it incorporates color resource editing similar to that of the contributed-software *xcol* program. There are several main categories of properties that you can edit:

Table 14-1. Property Categories

Category Name	Function	Restriction
Programs	Root Menu User Programs	AT&T-OL only
Color	Window, root window, data areacolors	Color monitor only

Table 14-1. Property Categories

Category Name	Function	Restriction
Icons	Icon placement	
Menus	Menu operations	OpenWindows only
Mouse Setting	Mouse speed, pointer jumping, etc.	
Mouse Modifiers	Mouse to OPEN LOOK mappings	AT&T-OL only
Keyboard Functions	Keyboard shortcuts	AT&T-OL only
Miscellaneous	Pointer focus, scrollbar side, etc.	

We'll discuss each of these in order, after showing you how to start up the properties editor.

14.1.1 Starting The Properties Editor

The Properties Editor, called the Workspace Manager Application in the AT&T documentation, can be started from the *Properties* entry in the *Workspace Menu* or root menu on either AT&T-OL or Sun OpenWindows.† The Sun OpenWindows Properties Editor can also be started from a shell window under the name *props*, i.e.,

```
props &
```

You should get an initial screen like Figure 14-1 , although the initial category will vary

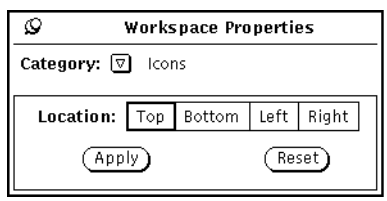


Figure 14-1. Properties Editor initial screen

from one release to the next. From here you can select any one of the categories by pulling down the abbreviated menu button, like so:

Note that on each category panel there is an *Apply* button. When you press this, two things happen. One is that your.*Xdefaults* file is rewritten; the other is that the new resource values are written as an X11 "property" on the root window, so that both *olwm* and any OPEN LOOK applications will notice them at once.

† On OpenWindows, it may not be available from the root menu if you have modified the menu file as described in Chapter 13, *Customizing olwm*.

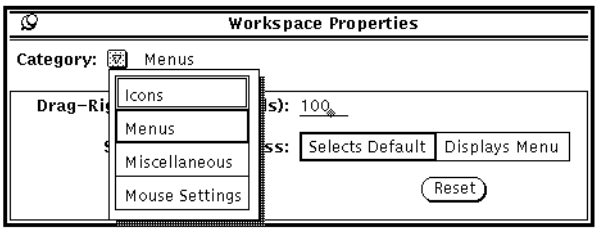


Figure 14-2. Abbreviated Menus on Properties.

The rewriting of the resource file *.Xdefaults* is not perfect. For one thing, the file is processed with the C language pre-processor as part of this re-writing, and a side effect of this is that any comments you have in it will be lost. Older OpenWindows versions therefore give you a Notice with this text:

```
Applying your changes will modify your ~/.Xdefaults file.
All comments in the file will be lost.
Do you want to do this?
```

The buttons on the Notice are *Yes* and *No*. If you click SELECT on *Yes*, your file will be rewritten, and (true to its word) all comments will be lost†. And the new properties will be applied to windows on the display. On the other hand, if you select *No*, no changes will be made, either to your file or to the root window property. If you don't do a lot of work on your *.Xdefaults* file, you may not care about comments in it.

Newer OpenWindowsversions of the Properties Editor instead update the *.OWdefaults* file in your home directory, which is normally loaded with *xrdb* as described in Chapter 12, *Setting Resources*.

Each panel also has a *Reset* button; note that this resets to your current value, not to the "factory default". The AT&T-OL version also has *Reset to Factory* on its categories, which resets the values in that category to the OPEN LOOK defaults.

You can select any of the categories for editing at any time. You don't have to *Apply* the changes from each panel; you only need to click the *Apply* button once before leaving the editor to save all your changes.

Notice also that the OpenWindows version comes up as a pinned, pop-up window, so it has the pop-up *Window* menu. To quit this program, you can either select *Dismiss* from the *Window* menu, or explicitly unpin-pin it by clicking SELECT on the pin to "pull the pin."

† You can fool this process by making your comments appear to be resources, as in:

```
comment.phony:this is a comment that will be preserved
```

14.1.2 Programs Menu Category

This category produces a scrolling list with the *Programs* category from the root menu. Note that if you have your own *.openwin-menu* file, it must include a file named *.openwin-menu-programs* from your home directory, or this category will be inoperative. For each item you can specify the menu label (such as *XTerm* and the command to be run (such as *xterm -fn fixed -bg pink*).

14.1.3 Color Property Category - OpenWindows

The next category is the Color Preferences menu. The OpenWindows version of the color window only appears if you are on a color display. Its appearance varies from one release of OpenWindows to another; an earlier version is shown in Figure 14-3 and the Version 3.3 window is shown in Figure 14-4 .

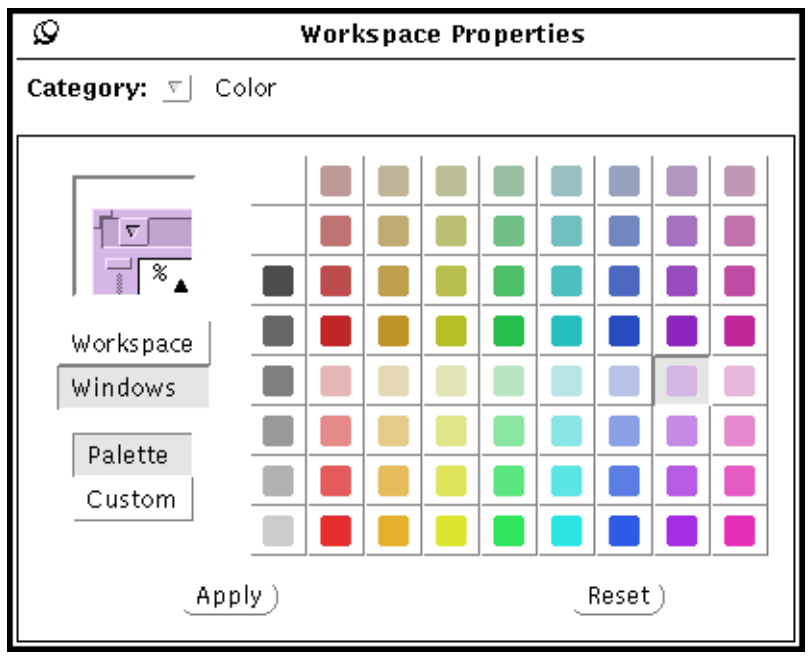


Figure 14-3. Color category - OpenWindows 3.1

The box at the center bottom (earlier versions had it in the top-left corner) of the Color category panel is a "model window" that shows a simulation of the top-left corner of a typical shell window under OPEN LOOK with the currently-selected colors. The exclusive-choice box underneath it lets you select between the OPEN LOOK Workspace (root window) color and the OPEN LOOK Window color. The "Window" color is used on window titlebars and on the background of controls such as scrollbars. The actual color of the text part of the window is not controlled by OPEN LOOK, but by the application, be it *cmdtool* or *xterm* or

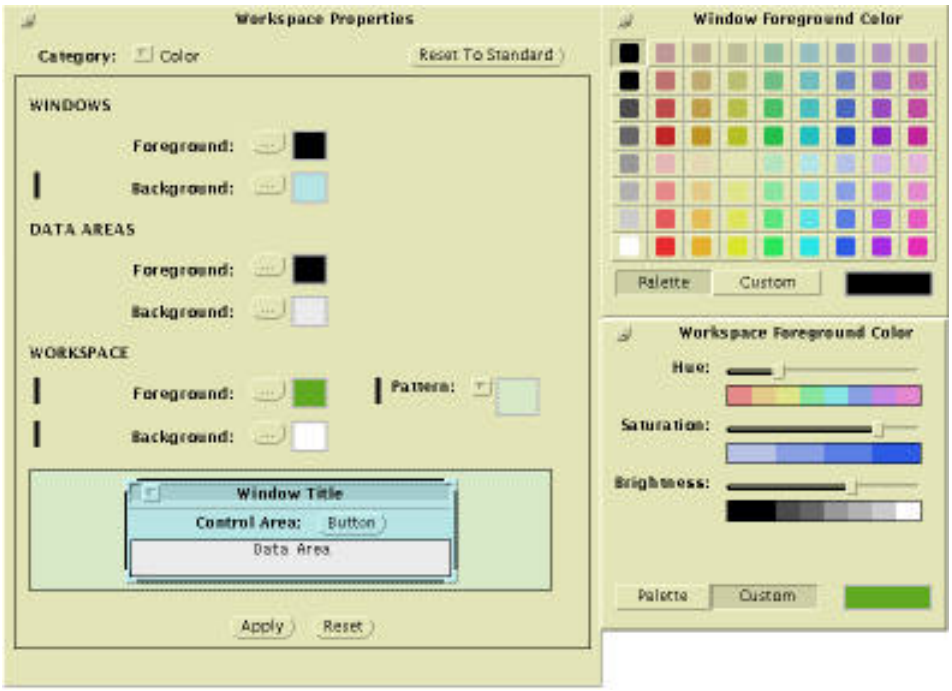


Figure 14-4. Color Category - OpenWindows 3.3

any other client. Having selected either *Workspace* or *Window* (or *Data Areas* in 3.3 or later), you can choose colors for it from among the dozens of colors displayed on the palette. Each time you pick a color, it is instantly shown on the appropriate parts of the "model window" in the upper left. Thus, you can see right away what your windows will look like if you select the *Apply* button.

You can probably find a color that you like from among the hundreds shown in the palette. But if not, don't despair. You can play artist and mix your own colors. Just select the *Custom* button, and the Palette will be replaced by a "mixer" window with sliders for Color, Saturation, and Brightness. It's difficult to describe these in a black-and-white book, so your best bet is to look at Figure 14-4 and experiment with these yourself.

Once you have the colors you like, click *Apply* to both update your *.Xresources* file and update the colors on your screen.

14.1.4 Icons Property Category

The *Icons* category lets you choose the location of icons, and in some cases the border display of icons. The screen looks like Figure 14-5.

Selecting one of the four locations will make that be the location for *olwm* to place icons. For example, *Top* means that icons will be placed in a row along the top edge of the screen.

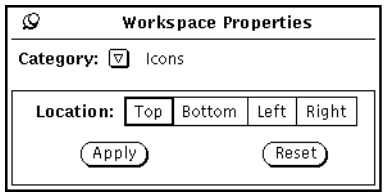


Figure 14-5. Icon Property Display.

The *Border* setting, which only shows up in the AT&T-OL version, specifies whether a border is displayed around each icon when you are on a color monitor. Under OpenWindows, you cannot control this setting. On a monochrome monitor, borders are always displayed.

As usual, click *Apply* to make your changes take effect.

14.1.5 Menus Property Category

This category only appears on the OpenWindows version. It contains only the items *Drag-Right distance* (which appears under *Mouse Settings* in the AT&T-OL version, and *SELECT Mouse Press* which appears under *Miscellaneous* category in AT&T-OL. To avoid duplication, the items are documented under the corresponding AT&T-OL entries.

14.1.6 Mouse Setting Property Category

This category has four items:

- *Multi-Click Timeout* specifies how close together two clicks of the same button can be and still be considered part of a single multi-click sequence instead of two separate clicks.
- *Mouse Damping* lets you filter out small mouse movements that may be accidental, to separate them from intentional motions of the mouse.
- *Drag-Right Distance* lets you specify how far you must drag the mouse right on a control with a menu mark in order to activate the submenu.
- *Menu Mark Region* Is the opposite; it specifies how far to the left you must move to get back to the menu mark, i.e., to pop down the submenu.

As usual, click *Apply* to make your changes take effect.

14.1.7 Miscellaneous Property Category

This is a kind of catch-all category, as you might expect by the name. It is a list of exclusive choices. The following lists the choices, their possible values, and which are currently available in the two OPEN LOOK implementations:

Table 14-2. Miscellaneous Properties

Name	Choices	Version
Beep	Always| Notices Only| Never	Both

Table 14-2. Miscellaneous Properties

Name	Choices	Version
Window layering	Individually| As A Group	AT&T-OL
Start OPEN LOOK at login	Yes| No	AT&T-OL
SELECT Mouse Press	Displays Default| Displays Menu	Note 1
Help Model	Input Focus| Pointer	AT&T-OL
Set Input Area	Click Select| Move pointer	Both
Interface Appearance	2D| 3D	AT&T-OL
Mnemonics	Off| Underline| Highlight| On-Don't Show	AT&T-OL
Accelerators	Off| On-Show| On-Don't Show	AT&T-OL
Scrollbar Placement	Left| Right	OpenWindows

Note 1: This item is in the *Miscellaneous* category in AT&T-OL, but is in the *Menus* category in OpenWindows.

14.1.7.1 Beep

Controls when the bell will ring: *Always* means whenever notices or important footer notices appear; *Notices Only* means only when a Notice appears, and *Never* means what it says.

14.1.7.2 Window layering Individually| As A Group

Controls how windows are moved to the front or back. *As A Group* means that all pop-ups move when the main window moves, while *Individually* treats each window as an independent item.

14.1.7.3 Start OPEN LOOK at login Yes|No

Controls whether the *olinit* script should be run from your *.profile* at login time.

14.1.7.4 SELECT Mouse Press Displays Default|Displays Menu

Controls what happens when you press SELECT on a control that has a menu mark. *Displays Default* means it activates the Default item for the menu. *Displays Menu* means that it pulls down or pops up the menu, but waits for you to select an item. *Displays Default* is the advanced user's choice, as it lets you get either the default (by clicking SELECT) or the menu (by clicking the MENU pointer button).

14.1.7.5 Help Model Input Focus|Pointer (AT&T-OL)

For programs written with the OLIT toolkit, controls whether HELP is offered for the text field that has the input focus, or the item that the pointer is over, when the HELP key is pressed.

14.1.7.6 Set Input Area Click Select|Move pointer

Selects the pointer focus policy, either click-to-type or pointer focus. See the discussion of pointer focus policy in Chapter 1, *An Introduction to OPEN LOOK and the X Window System*.

14.1.7.7 Interface Appearance 2D|3D

Specifies whether to use the two-dimensional or three-dimensional appearance. Monochrome monitors always use 2D; color monitors default to 3D but can be controlled to 2D by this setting.

14.1.7.8 Mnemonics Off|Underline|Highlight|On-Don't' Show

AT&T-OL lets you use single-character mnemonics for many functions. This item controls whether the mnemonics will be disabled, shown by underlining or highlighting the letter, or be enabled but not displayed.

14.1.7.9 Accelerators Off|On-Show|On-Don't Show

Similarly for keyboard accelerators.

14.1.7.10 Scrollbar Placement Left|Right

Controls whether vertical scrollbars appear to the left or right of the panel they are controlling.

Don't forget to click *Apply* to make your changes take effect.

This concludes our discussion of the OPEN LOOK-specific Properties Editor or Workspace Manager. We now turn our attention to some standard X11 customization clients that will be present in any standard version of The X Window System.

14.2 xset: Setting Display and Keyboard Preferences

The *xset* client allows you to set an assortment of user preference options for the display and keyboard. Some of these are followed by `on` or `off` to set or unset the option. Note that *xset* is inconsistent with other UNIX and X programs in its use of a dash (–) as an option flag. Some options use a preceding dash to indicate that a feature be disabled; this can be confusing at first to users accustomed to seeing a dash as an introductory symbol on all options.

Although *xset* can be run any time, it is suggested that you run it at startup. These settings reset to the default values when you log out. Not all X implementations honor all of these options.

14.2.0.1 Keyboard Bell

The `b` option controls bell volume (as a percentage of its maximum), pitch (in hertz), and duration (in milliseconds). It accepts up to three numerical parameters:

`b` ***`volume pitch duration`***

If no parameters are given, the system defaults are used. If only one parameter is given, the bell volume is set to that value. If two values are listed, the second parameter specifies the bell pitch. If three values are listed, the third one specifies the duration.

For example, the command:

```
% xset b 70 1000 100
```

sets the volume of the keyboard bell to 70 percent of the maximum, the pitch to 1000 hertz, and the duration to 100 milliseconds.

Note that bell characteristics vary with different hardware. The X server sets the characteristics of the bell as closely as it can to the user's specifications.

The `b` option also accepts the parameters `on` or `off`. If you specify `xset b on`, system defaults for volume, pitch and duration are used.

The bell can also be turned off with the option `-b`, or by setting the volume parameter to 0 (`xset b 0`).

14.2.0.2 Bug Compatibility Mode

Some X11 Release 3 clients were written to work with "features" of the MIT X11 Release 3 server, features which could more accurately be called bugs. Many of these bugs have been eliminated from the MIT Release 4 server. In order to allow certain Release 3 clients to work under the Release 4 server, the MIT X11 Release 4 server has a bug compatibility mode that can be enabled using *xset*. In this mode, the Release 4 server is compatible with Release 3 clients that depended on bugs in the Release 3 server to work properly (most notably the Release 3 version of *xterm*). This feature is hardly ever needed today; newer servers no longer provide this compatibility, and tend to ignore requests for it from *xset*.

Note that since the OpenWindows server is not derived from the MIT Sample Server, this option is **not** available under OpenWindows. However, the XWIN server is an enhanced version of the MIT server, so this option does work on the AT&T server.

To enable bug compatibility mode, use the command `xset bc`; to disable it, use the command `xset -bc`.

14.2.0.3 Keyclick Volume

The `c` option sets the volume of the keyboard's keyclick and takes the form:

`c` ***`volume`***

`volume` can be a value from 0 to 100, indicating a percentage of the maximum volume. For example:

```
% xset c 75
```

sets a moderately loud keyclick. The X server sets the volume to the nearest value that the hardware can support.

The `c` option also accepts the parameters `on` or `off`. If you specify `xset c on`, the system default for volume is used.

The keyclick can also be turned off with the option `-c`, or by setting the volume parameter to 0 (`xset c 0`).

On some hardware, a volume of 0 to 50 turns the keyclick off, and a volume of 51 to 100 turns the keyclick on.

14.2.0.4 Enabling or Disabling Auto-repeat

The `r` option controls the keyboard's auto-repeat feature. (Auto-repeat causes a keystroke to be repeated over and over when the key is held down.) Use `xset r` or `xset r on` to enable key repeat. Use `xset -r` or `xset r off` to disable key repeat. On some keyboards (notably Apollo) only some keys repeat regardless of the state of this option.

14.2.0.5 Changing or Rehashing the Font Path

As discussed in Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, when a client is to be displayed in a particular font, the server by default looks for the font in several subdirectories of */usr/lib/X11/fonts*, such as *misc*, *75dpi*, and *100dpi*.

The `fp` (font path) option of *xset* can be used to change the font path, i.e., to direct the X server to search other directories for fonts called by a client. The option must be followed by a directory or a comma-separated list of directories, as in this example:

```
% xset fp /work/andy/fonts,/usr/lib/X11/newfonts
```

To restore the server-specific default font path, type:

```
% xset fp default
```

The `fp` option with the `rehash` parameter causes the server to reread the *fonts.dir* and *fonts.alias* files in each directory in the current font path. You need to do this every time you edit an alias file to make the server aware of the changes. To make the server aware of aliases, type:

```
% xset fp rehash
```

You also have to do this if you add or remove fonts. See the *xset* manual page for more information.

14.2.0.6 Keyboard LEDs

The `led` option controls the enabling or disabling of one or e fall of the keyboard's LEDs. It accepts the parameters `on` or `off` to enable or disable all of the LEDs. A preceding dash also disables all of the LEDs (`-led`).

You can also enable or disable individual LEDs by supplying a numerical parameter (a value between 1 and 32) that corresponds to a particular LED. The `led` option followed by a numerical parameter enables that LED. The `led` option preceded by a dash and followed by a numerical parameter disables that LED. For example:

```
% xset led 3
```

would enable LED #3, while:

```
% xset -led 3
```

would disable LED #3.

Note that the particular LED values may refer to different LEDs on different hardware.

14.2.0.7 Pointer Acceleration

The `m` (mouse) option controls the rate at which the mouse or pointer moves across the screen. This option takes two parameters: ***`acceleration`*** and ***`threshold`***. They must be positive integers. (The acceleration can also be written as a numerator/denominator combination separated by a slash, for example, 5/4.)

The mouse or pointer moves ***`acceleration`*** times as fast when it travels more than the ***`threshold`*** number of pixels in a short time. This way, the pointer can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen by a flick of the wrist when desired. If only one parameter is given, it is interpreted as the acceleration.

For example, the command:

```
% xset m 5 10
```

sets the pointer movement so that if you move the pointer more than 10 pixels, the pointer cursor moves five times as many pixels on the screen as you moved the pointer on the pad.

If no parameter or the value `default` is used, the system defaults will be set.

If you want to change the threshold and leave the acceleration unchanged, enter the value `default` for acceleration.

14.2.0.8 Screen Saver

X supports a screen saver to blank or randomly change the screen when the system is left unattended for an extended period. This avoids the “burn in” that can occur when the same image is displayed on the screen for a long time. The `s` (screen saver) option to *xset* determines how long the server must be inactive before the screen saver is started.

The `s` option takes two parameters: ***`time`*** and ***`cycle`***. The screen goes blank if the server has not received any input for the time interval specified by the ***`time`*** parameter. The contents of the screen reappear upon receipt of any input. If the display is not capable of blanking the screen, then the screen is shifted a pixel in a random direction at time intervals set by the ***`cycle`*** parameter. The parameters are specified in seconds.

For example, the command:

```
% xset s 600
```

sets the length of time before the screen saver is invoked to 600 seconds (10 minutes).

For a display not capable of blanking the screen, the command:

```
% xset s 600 10
```

sets the length of time before the screen saver is invoked to 10 minutes and shifts the screen every 10 seconds thereafter, until input is received.

The `s` option also takes the parameters:

`default` Resets the screen save option to the default.

`blank` Turns on blanking and overrides any previous settings.

`noblank` Displays a background pattern rather than blanking the screen; overrides any previous settings.

`off` Turns off the screen saver option and overrides any previous settings.

`expose` Allows window exposures (the server can discard window contents).

`noexpose` Disables screen saver unless the server can regenerate the screens without causing exposure events (i.e., without forcing the applications to regenerate their own windows).

OpenWindows Option—Sun users can bypass this, and instead start the program */usr/bin/screenblank* from */etc/rc.local* during system boot. This screen blank program has the advantage over the X server's that it works all the time, not just while X11 (OpenWindows) is running. For example, it will work before you have logged in, or while you are logged in but not running any window system.

14.2.0.9 Color Definition

On color displays, every time a client requests a private read/write colorcell, a new color definition is entered in the display's colormap. The `p` option sets one of these colormap entries even though they are supposed to be private. The parameters are a positive integer identifying a cell in the colormap to be changed and a color name:

```
p entry_number color_name
```

The root window colors can be changed on some servers using *xsetroot*. An error results if the map entry is a read-only color.

For example, the command:

```
% xset p 3 blue
```

sets the third cell in the colormap to the color blue but only if some client has allocated this cell read/write.

The client that allocated the cell is likely to change it again sometime after you try to set it, since this is the usual procedure for allocating a read/write cell.

14.2.0.10 Help with xset Options

The `q` option lists the current values of all *xset* preferences.

14.3 xsetroot: Setting Root Window Characteristics

You can use the *xsetroot* client to tailor the appearance of the background (root) window on a display running X.

The *xsetroot* client is primarily used to specify the root window pattern: as a plaid-like grid, tiled gray pattern, solid color, or a bitmap. You can also specify foreground and background colors (defaults are black and white), reverse video, and set the shape of the pointer when it's in the root window.

If no options are specified, or the `-def` option is specified, *xsetroot* resets the root window to its default state, a gray mesh pattern, and resets the pointer to the hollow X pointer. The `-def` option can also be specified with other options; those characteristics that are not set by other options are reset to the defaults.

Although *xsetroot* can be run at any time, it is suggested that you run it from a startup shell script, as described at the end of this chapter. All settings reset to the default values when you log out.

For a complete list of options, see the *xsetroot* reference page in Part Three of this guide. Not all X implementations are guaranteed to support all of these options. Some of the options may not work on certain hardware devices.

The `-help` option prints all the *xsetroot* options to standard output. The options you'll probably use most frequently are explained in the next section. Since only one type of background pattern can be specified at a time, the `-solid`, `-gray`, `-grey`, `-bitmap` and `-mod` options are mutually exclusive.

14.3.1 Setting Root Window Patterns

The default root window pattern is called a "gray mesh." On most displays, it is fairly dark.

The *xsetroot* client allows you to specify an alternative gray background with the `-grey` (or `-gray`) option. This tiled gray pattern is slightly lighter than the default gray mesh pattern.

The *xsetroot* client also allows you to create a root window made up of repeated "tiles" of a particular bitmap, using the option:

```
-bitmap filename
```

where ***filename*** is the bitmap file to be used as the window pattern.

You can choose any of the bitmaps in the directory */usr/include/X11/bitmaps* or make your own bitmap files using the *iconedit* or *bitmap* client (see Chapter 9, *Graphics Clients*).For example, the command:

```
% xsetroot -bitmap /usr/openwin/share/include/Xol/bitmaps/gumby \
    -fg white -bg skyblue
```

fills the root window with a tiling of the bitmap */usr/openwin/share/include/Xol bitmaps/skyscene*, a tranquil scene, using the colors white and blue.

The `-mod` option sets a plaid-like grid pattern on the root window. You specify the horizontal (x) and vertical (y) dimensions in pixels of each square in the grid. The syntax of the option is:

```
-mod x y
```

where the parameters ***x*** and ***y*** are integers ranging from 1 to 16 (pixels). (Zero and negative numbers are taken as 1.)

The larger the x and y values you specify, the larger (and more visible) each square on the root window grid pattern. Try the command:

```
% xsetroot -mod 16 16
```

for the largest possible grid squares. Then test different x and y specifications.

The *xsetroot* option:

```
-solid color
```

sets the color of the root window to a solid color. This can be a color from the color name database or a more exact color name specified by its RGB value.

The command:

```
% xsetroot -solid lightblue
```

sets the color of the root window to light blue.* See Chapter 11, *Command Line Options*, and Chapter 12, *Setting Resources*, for more information on how to specify colors.†

While this behavior may seem to be a serious bug, it is actually an optimization designed to ensure applications don't run out of colors unnecessarily. Free colormap cells can be a scarce resource. See Volume One, *Xlib Programming Manual*, for more information.

14.3.2 Foreground Color, Background Color, and Reverse Video

In addition to specifying a solid color for the root window pattern, *xsetroot* allows you to specify foreground and background colors if you set the pattern with `-bitmap` or `-mod`. The standard Toolkit options are used to set foreground and background colors: `-fg` and `-bg`. The defaults are black and white.

Colors can be specified as names from the color name database, or as RGB values. See Chapter 11, *Command Line Options*, and Chapter 12, *Setting Resources*, for more instructions on how to specify color.

If you specify reverse video (`-rv`), the foreground and background colors are reversed.

† For technical reasons, colors set with `xsetroot -solid` may change unexpectedly. When you set a color with the `-solid` option to *xsetroot*, the client allocates a colorcell, sets the color, and deallocates the colorcell. The root window changes to that color. If another client is started that sets a new color, it allocates the next available colorcell—which may be the same one xsetroot just deallocated. This results in that color changing to the new color. The root window also changes to the new color. If this happens, you can run *xsetroot* again and if there are other colorcells available, the root window changes to the new color. If all colorcells are allocated, any call to change a colorcell results in an error message.

Foreground and background colors also take effect when you set the root window pointer, as described in the next section.

Another use of *xsetroot* is to notify you when a long-running program is done. For example, if you expect a program compilation being run by *make* to take a while, you can "type ahead" an *xsetroot* command. For example:

```
make
xsetroot -solid pink
```

then iconify the window. It will run the *make*, then change your background window pink—a change you are likely to notice.

14.3.3 Changing the Root Window Pointer

By default, the pointer is an X when it's in the root window. You can change the shape of the root window pointer to one of the standard X cursor shapes or to any bitmap, using these options:

```
-cursor_name standard_cursor_name
-cursor cursorfile maskfile
```

Available as of MIT X11 Release 4, the first option allows you to set the root window pointer to one of the standard cursor symbols, which are generally listed in the file */usr/include/X11/cursorfont.h*. We've provided a list of the standard cursors in Appendix D, *Standard Cursors - X11 and OPEN LOOK*. To specify a standard cursor on a command line or in a resource file, strip the XC_ prefix from the name. Thus, to set the root window pointer to the pirate cursor symbol, you would enter:

```
% xsetroot -cursor_name pirate
```

This second option is intended to allow you to set the root window pointer to a bitmap, perhaps one you create. The parameters ***cursorfile*** and ***maskfile*** are bitmaps. The ***cursorfile*** sets the bitmap for the pointer shape. In effect, the ***maskfile*** is placed behind the ***cursorfile*** bitmap to set it off from the root window. The ***maskfile*** should be the same shape as the ***cursorfile*** but should generally be at least one pixel wider in all directions.†

With the *xsetroot* defaults, you can observe the effect of a mask. When you move the X pointer onto the dark gray root window, the X should have a very thin white border, which enables you to see it more clearly.

For the ***cursorfile***, you can use any of the standard bitmaps in */usr/include/X11/bitmaps* or you can make your own with the *bitmap* client (see Chapter 9, *Graphics Clients*).

Every standard cursor has an associated mask. Pictures of the cursors appear in Appendix D, *Standard Cursors - X11 and OPEN LOOK*. To get an idea of what masks look like, display the cursor font using the command:

† Technically speaking, the mask determines the pixels on the screen that are disturbed by the cursor. It functions as a sort of outliner or highlighter for the cursor shape. The mask appears as a white (or background color) border around the cursor (black or another foreground color), making it visible over any root window pattern. This is especially important when a black cursor appears on a black root window.

```
% xfd -fn cursor.
```

If you are using your own bitmap as the ***cursorfile***, until you get used to the way masks work, create a ***maskfile*** that is a copy of the ***cursorfile*** with all bits set, i.e., the ***maskfile*** should be all black* (or the foreground color). Then edit the ***maskfile*** to make it wider than the ***cursorfile*** by at least one pixel in all directions.†

To specify a root window pointer made from the smiling Gumby bitmap we created for Figure 7-2, first copy the bitmap to make a mask file:

```
% cp gumby gumby.mask
```

Then edit the *gumby.mask* file using the *bitmap* client, setting all squares inside the Gumby. (You can use the *bitmap* command box **Flood Fill** to set all the empty squares at once.) Continue to edit the bitmap, making it one pixel wider in all directions.

Then specify the new pointer with *xsetroot*:

```
% xsetroot -cursor gumby gumby.mask
```

See Chapter 9, *Graphics Clients*, for more information on using *bitmap*.

14.4 xmodmap: Modifier Key and Pointer Customization

The *xmodmap* client is used to assign (or map) key functions to physical keys on the keyboard. Primarily, *xmodmap* is used to assign so-called "modifier" key functions to physical keys but it can also change the way other keys (and even pointer buttons) function.

As described in , keys with labels such as Shift, Control, Caps Lock, etc. are called "modifier" keys because they modify the action of other keys. The number and names of modifier keys differ from workstation to workstation. Every keyboard is likely to have a Shift, Caps Lock, and Control key but after that, the babble begins. One workstation might have an Alt key, another might have a Funct key, and yet another a Gold key. On the Sun-3 keyboard, there are no less than three additional modifier keys, labeled Alternate, Right, and Left.

Because of the differences between keyboards, X programs are designed to work with *logical* modifier keynames. The logical keynames represent functions recognized by X programs. These modifier keynames can be mapped by the user to any physical key on the keyboard with the *xmodmap* client.

The logical keynames that X recognizes are:

- Shift
- Lock

† Don't be confused by the idea of a black cursor with a black mask on a black root window. Remember, the mask determines the pixels that are disturbed by the cursor—in effect creating an outline around the cursor. The outline appears in white (or specified background color), regardless of the color of the ***maskfile***.

- Control
- Mod1 (Meta or Alt)
- Mod2
- Mod3
- Mod4
- Mod5

These keynames are case-insensitive.

Of these X modifier keys, only Shift, Caps Lock, Control, and Meta are in common use. Note that *uwm* also recognizes the mod keys simply by number alone (1-5) and recognizes mod1 as Meta (i.e., mod1, Meta and 1 are equivalent).

The primary function of *xmodmap* is to allow you to assign these important modifier keyname functions (Shift, Control, Meta, etc.) to convenient keys on the keyboard. For example, you could choose to map the Shift function to a single key called "Shift," to two "Shift" keys (one on either side of the keypad), to an "Alt" key, or to any other convenient key or keys on the physical keyboard. A left-handed person might choose to map modifier keys that more often are found on the left side, such as Control, on the right side of the keyboard.

In practical terms, each server will have a default keyboard configuration. The Shift, Caps Lock, and Control modifier keynames will be mapped to obvious keys. The assignment of the Meta key might be less obvious.

The *xmodmap* client allows you to print out the current assignments of modifier keyname functions to physical keys and/or to change the assignments.

xmodmap also has two other functions that you will probably use less frequently. In addition to mapping modifier keyname functions to physical keys, *xmodmap* also allows you to assign the function of *any* key on the keyboard to any other key. For instance, you can make the Backspace key and the Delete key both function as Delete keys. (This may be helpful if the Backspace key is easier to reach.)

Also, in addition to keyboard mappings, *xmodmap* can be used to display or change the pointer button assignments. Many X clients recognize logical pointer button commands. For example, holding down and dragging the first logical pointer button in an *xterm* window copies the text into memory. (In many default pointer maps, the first logical button is the leftmost button, designed to be pressed by the right index finger.) Each logical button is associated with a *button code*. The first logical button generates button code 1, the second logical button generates button code 2, etc. *xmodmap* allows you to reassign logical buttons to different physical buttons on the pointer.

Thus, basically, *xmodmap* can perform three types of mappings:

1. Assign modifier keyname functions (such as Shift, Control, Meta) recognized by X to physical keys.
2. Make any key on the keyboard function as any other key (for example, making Backspace function like Delete).
3. Reassign logical pointer button functions to other physical buttons (for example, making the rightmost physical button function as the first logical button).

In the following sections, we discuss key mapping, with an emphasis on the first type of mapping, of modifier keyname functions. Chances are, you'll have relatively little call to map other key functions (such as Backspace), though we have included an example of one such mapping, just in case.

After considering key mapping, we'll take a look at the much simpler issues involved in mapping pointer button functions. As you might expect, when you're changing the functionality of (up to) three pointer buttons, it's fairly simple to keep track of what you're doing.

On the other hand, mapping modifier key functions to physical keys can be more than a little confusing. In order to understand the mechanics of mapping keys, we first need to take a look at some terms used to describe keyboard keys.

14.4.1 Keycodes and Keysyms

Each key on a physical keyboard can be identified by a number known as a *keycode*. (Technically speaking, a keycode is the actual value that the key generates.) Keycodes cannot be mapped to other keys. No matter what functions you assign to various keys with *xmodmap*, the keycode associated with each physical key remains the same.

In addition to a keycode, each physical key is associated with a name known as a keysym. A *keysym* (*key sym*bol name) is a name that represents the label on a key (theoretically) and corresponds to its function.

Alphanumeric keys generally have obvious keysyms, corresponding to the label on the key: for example, the keysym for the key labeled "H" is *h*. Unfortunately, a keysym does not always correspond to the key label. For example, on a Sun-3 workstation, though the keysym for the key labeled "Return" is *Return*, the keysym for the key labeled "Alternate" is *Break*, and the keysym for the key labeled "Right" is *Meta_R*.

While each keycode is tied to a physical key, each keysym corresponds to a *function* —and the keysym/function is mapped to a particular physical key (keycode). Every keyboard has a default assignment of keysyms to keycodes. In most cases, each physical key on the keyboard will be associated with a different keysym. As we'll see, however, the keysym (function) associated with a particular physical key (keycode) can be changed. This is done by assigning the keysym of one key to the keycode of another.

The modifier keynames recognized by X are not to be confused with keysyms. The X modifier keys are limited to the eight keynames discussed previously and are assigned *in addition* to the regular keysym/keycode pairings. In other words, when a physical key is

mapped to function as the X Control key, it already has a default functionality (keysym) and keycode.

By default, most modifier keyname functions are mapped to keys having keysyms representing the same function. For example, the X Control keyname is probably mapped to the key labeled Control, and having the keysym Control.

The Meta modifier keyname is probably also assigned to a key having the keysym Meta. However, determining which physical key has the keysym Meta can be something of a puzzle. Later in this chapter, we'll consider a program called *xev*, which can be used to determine the keysym and keycode of any physical key.

With this background information in mind, we can now tackle a procedure to map modifier keynames.

14.4.2 Procedure to Map Modifier Keys

In order to change modifier key mappings with a minimum of confusion, you should perform these steps:

1. Display the current *modifier* key mappings using *xmodmap.*
2. Then print out the default assignments of keysyms to keycodes for *all* keys, using *xmodmap* with the `-pk` option. Save this list of the default key assignments in a file as a reference.
3. Experiment with the *xev* client to determine the keysyms associated with certain physical keys. This will help you find the key(s) assigned as the Meta modifier key (which probably also has the keysym Meta).
4. Once you're familiar with the current assignments, you can remap modifier keys using *xmodmap.*

14.4.3 Displaying the Current Modifier Key Map

Before mapping any modifier keynames, you should take a look at the current assignments. With no options, *xmodmap* displays the current map of X modifier keynames to actual keys. Type *xmodmap* and you get a display similar to this:

```
xmodmap: up to 3 keys per modifier, (keycodes in parentheses):

shift Shift_L (0x6a), Shift_R (0x75)
lock Caps_Lock (0x7e)
control Control_L (0x53)
mod1 Meta_L (0x7f), Meta_R (0x81)
mod2 Mode_switch (0x14)
mod3 Num_Lock (0x69)
mod4 Alt_L (0x1a)
mod5 F13 (0x20), F18 (0x50), F20 (0x68)
```

For each logical keyname (on the left), *xmodmap* lists one or more keysyms, each followed in parentheses by an actual hardware keycode. The keycodes displayed by *xmodmap* are

represented in hex. As we'll see, the equivalent decimal and octal keycodes are also accepted as arguments to *xmodmap*.

Table 14-3. Arguments to xmodmap

Logical modifier keyname recognized by X	**Keysym**	**Keycode (hex version)**
Shift	Shift_L Shift_R	(0x6a) (0x75)
Lock	Caps_Lock	(0x7e)
Control	Control_L	(0x53)
Mod1	Meta_L Meta_R	(0x7f) (0x81)

In this mapping, two keys are assigned as Meta (mod1) keys: keys having the keysyms Meta_L and Meta_R (for left and right, apparently one on each side of the keyboard). Unfortunately, as you can see, this doesn't really tell you which keys these are on the physical keyboard. You still need to know which physical keys (keycodes) have the keysyms Meta_L and Meta_R. You can determine this using the *xev* client, described later in this chapter.

14.4.4 Determining the Default Key Mappings

Before you start mapping keys, you should display and save a map of the default assignments of keysyms to keycodes. Running *xmodmap* with the `-pk` option prints a current map of all keyboard keys to standard output. This map, called a keymap table, lists the decimal keycode on the left and the associated keysym(s) on the right. Figure 14-6 shows a portion of a typical keymap table, for a Sun-4 keyboard.

Notice that each keysym is listed by a keysym name (comma, Caps_Lock, etc.) and a keysym value (0x002c, 0xffe5, etc). For our purposes, this value is irrelevant. It cannot be supplied as a keysym argument to *xmodmap*.

As you can see, the keymap table lists regular keyboard keys (C, V, comma, slash, space, etc.), and function/numeric keypad keys (R13, F35, etc.) as well as modifier keys (Caps_Lock, Meta_L and Meta_R). If you map several keys, you may get confused as to the original assignments. Before you map any keys, we suggest you redirect the keymap table to a file to save and use as a reference:

```
% xmodmap -pk > keytable
```

The keysyms recognized by your server are a subset of a far greater number of keysyms recognized internationally. The file */usr/include/X11/keysym.h* lists the keysym *families* that are enabled for your server. The file */usr/include/X11/keysymdef.h* lists the keysyms in each of the families enabled for your server, as well as the keysyms in several other fami-

lies. See Appendix H, *Keysyms*, of Volume Two, *Xlib Reference Manual*, for more information on keysyms and tables of the most common ones.

```
% xmodmap -pk
There are 4 KeySyms per KeyCode; KeyCodes range from 8 to 132.

 KeyCode Keysym (Keysym) ...
 Value Value (Name) ...

 109 0x0043 (C)
 110 0x0056 (V)
 111 0x0042 (B)
 112 0x004e (N)
 113 0x004d (M)
 114 0x002c (comma) 0x003c (less)
 115 0x002e (period) 0x003e (greater)
 116 0x002f (slash) 0x003f (question)
 117 0xffe2 (Shift_R)
 118 0xff0a (Linefeed)
 119 0xffde (R13) 0xffde (R13) 0xffb1 (KP_1) 0xff57 (End)
 120 0xff54 (Down) 0xffdf (F34) 0xffb2 (KP_2)
 121 0xffe0 (F35) 0xffe0 (F35) 0xffb3 (KP_3) 0xff56 (Next)
  . . .
 125 0xff6a (Help)
 126 0xffe5 (Caps_Lock)
 127 0xffe7 (Meta_L)
 128 0x0020 (space)
 129 0xffe8 (Meta_R)
```

Figure 14-6. Partial keymap table

14.4.5 Matching Keysyms with Physical Keys Using xev

The keysym and keycode for any key can be determined with the *xev* client.†

If you cannot use *xev*, you must rely on the keymap table and a little deductive reasoning. Since certain OPEN LOOK functions have keyboard shortcuts involving the Alt (Meta) key, at least on the AT&T version of OPEN LOOK, testing these shortcuts should help you locate this key. See , for more information.

This is particularly useful for finding the Meta key(s). The *xev* client is used to keep track of *events*, packets of information that are generated by the server when actions occur and are interpreted by other clients. Moving the pointer or pressing a keyboard key cause input events to occur. (For more information about events, see Volume One, *Xlib Programming Manual*.)

To use *xev*, enter the command:

```
% xev
```

† *xev* is an MIT X11 Release 3 standard client. In Release 4, it has been moved to the *demos* directory; on OpenWindows it is in */usr/openwin/demo/xev*. If an executable version does not exist on your system, ask your system administrator.

in an *xterm* window, and then use the pointer to place the *xev* window, as in Figure 14-7.

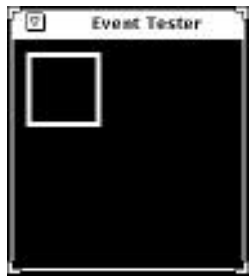


Figure 14-7. xev window

Within the *xev* window is a small box. Move the pointer inside this box. When you type a key inside the box, information about the key, including its keysym and keycode, will be displayed in the *xterm* window from which you started *xev*. The relevant information will look like this:

```
. . . keycode 127 (keysym 0xffe7, Meta_L) . . .
```

Notice that the keycode is given as a decimal number. You can use the decimal keycode as an argument to *xmodmap*. The keysym is listed by name, Meta_L, and value, 0xffe7. Again, this value cannot be supplied as a keysym argument to *xmodmap*. (See the *xev* reference page in Part Three of this guide for more information.)

To find the Meta key, type a few likely keys in the *xev* window. To terminate the program, you can use the *Window* menu or (if you didn't background the *xev*) just type Control-C in the window from which you invoked *xev*.

14.4.6 Changing the Map with xmodmap

xmodmap executes an expression or list of expressions that is interpreted as instructions to modify the key (or pointer) map. The expressions that can be interpreted by *xmodmap* are described in the next section.

xmodmap has this syntax:

xmodmap [***options***] [***filename***]

An expression can be executed in either one of two ways:

- From the command line, using the `-e` ***expression*** option. This option specifies an expression to be executed (as an instruction to modify the map). Any number of expressions may be specified from the command line. An ***expression*** should be enclosed in quotes.
- Entered in a file that is used as an argument to *xmodmap*. Several expressions can be entered in one file.

See the *xmodmap* reference page in Part Three of this guide for a complete list of options. Other than `-e` ***`expression`***, the most important options for our purposes are listed below.

`-n` Indicates that *xmodmap* should not change the key mappings as specified in the ***`filename`*** or command line expression but should display what it would do. A handy test. (Only works with key mappings, not with expressions that change the pointer map.)

`-verbose` Indicates that *xmodmap* should print information as it parses its input.

`filename` specifies a file containing *xmodmap* expressions to be executed (as instructions to modify the map). This file is usually kept in the user's home directory with a name like *.xmodmaprc*.

14.4.6.1 Expressions to Change the Key Map

The expressions interpreted by *xmodmap* can be used to perform these types of key mappings:†

1. Assign and remove keysyms as modifier keynames recognized by X.
2. Map any keysym (function) to any physical key (keycode).

This list shows allowable expressions, divided by function. (Using *xmodmap* with the `-grammar` option returns a help message with much of this information.) Those expressions that include an equal sign require a space before and after the sign.

1. To assign and remove keysyms as modifier keynames:

`clear` ***`MODIFIERNAME`***

Removes all entries in the modifier map for the given modifier, where valid modifier names are: shift, lock, control, mod1, mod2, mod3, mod4, and mod5 (case does not matter in modifier names, although it does matter for all other names). For example, the expression `clear Lock` will remove all keys that were bound to the lock modifier.

`add` ***`MODIFIERNAME=KEYSYMNAME`***

Adds the given keysym to the indicated modifier map. For example, you could make the Alt key an additional shift modifier key. The keysym name is evaluated after all input expressions are read to make it easy to write expressions to swap keys.

`remove` ***`MODIFIERNAME=KEYSYMNAME`***

Removes the given keysym from the indicated modifier map (unmaps it). For example, remove Caps_Lock as the lock modifier key. Unlike with the `add` expression, the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether they have been reassigned.

† Expressions to change the pointer map are discussed in the section "Displaying and Changing the Pointer Map," later in this chapter.

2. To map any keysym(s) to any physical key (keycode):

`keycode` ***NUMBER=KEYSYMNAME***

Assigns the keysym to the indicated keycode (which may be specified in decimal, hex or octal). Usually only one keysym is assigned to a given code.

`keysym` ***KEYSYMNAME=KEYSYMNAME***

Assigns the keysym on the right to the keycode of the keysym on the left. Note that if you have the same keysym bound to multiple keys, this might not work.

14.4.6.2 Key Mapping Examples

Expressions can be used on the *xmodmap* command line or entered in a file that is then used as an argument to *xmodmap*. Note that *xmodmap* should be run from your startup script (discussed in) to take effect for all clients in the login session. This section includes three examples, corresponding to the three types of mappings you can perform.

Remember that including the `-n` option on the *xmodmap* command line allows you to see what the new mappings *would* be, without actually performing them. This can be very useful, particularly while you're learning to use *xmodmap* and getting used to the syntax of expressions. (Note, however, that `-n` cannot be used with expressions to change the pointer mapping.)

First, the *xmodmap* client also allows you to assign logical modifier keynames to physical keys. A not so obvious feature of *xmodmap* is that to change the mapping of a modifier key, you must first remove that key from the current modifier map. For example, to swap the left Control and (Caps) Lock keys, you would first need to unmap both physical keys (Caps_Lock, Control_L) from their respective modifier keynames (lock, control):

```
remove lock = Caps_Lock remove control = Control_L
```

And then reverse the mappings:

```
add lock = Control_L add control = Caps_Lock
```

If you then type *xmodmap* without options, you see the new map:

```
xmodmap: up to 2 keys per modifier, (keycodes in parentheses):
shift Shift_L (0x6a), Shift_R (0x75)
lock Control_L (0x53)
control Caps_Lock (0x7e)
mod1 Meta_L (0x7f), Meta_R (0x81)
mod2
mod3
mod4
mod5
```

The key with the keysym Control_L functions as a Lock key and the key with the keysym Caps_Lock functions as a Control key.

Second, *xmodmap* allows you to assign any keysym to any other key. Here is some code that sets up the OpenWindows key assignments:

```
xmodmap -e 'keysym F1 = Help' \
    -e 'add mod1 = Meta_L Meta_R'
```

This sets F1 to be the OPEN LOOK Help key, and the Left and Right keys to be Meta (or ALT) keys. This example is excerpted from the Sun startup scripts in */usr/openwin/lib*. As another example, you might make the Backspace key function as a Delete key:

```
% xmodmap -e 'keysym BackSpace = Delete'
```

Then when you display the keymap table and *grep* for the Delete keysym, you'll see that it is assigned twice. On the command line of an *xterm* window, type:

```
% xmodmap -pk | grep Delete
```

and you'll get two lines from the current keymap table, similar to these:

```
50 0xffff (Delete)
73 0xffff (Delete)
```

The 50 and 73 are keycodes representing two physical keys. As you can see, both of these keys now function as Delete keys.

This example suggests some of the confusion you can experience using *xmodmap*. We know that one of these keys previously functioned as the Backspace key. But how can we tell which one? Here is an instance when our default keymap table comes in handy. If you've run `xmodmap -pk` and redirected it to a file before changing any mappings, you can check the file for the keysyms originally associated with the keycodes 50 and 73. In this case, the file tells us 50 originally was Backspace and 73 was Delete.

Of course, you could also figure out the original assignments by remapping one of the keycodes to Backspace. Then, if the key marked Backspace functions as marked, you know you've mapped the keysym to the original keycode. But, as you can see, the default keymap table can greatly simplify matters.

This example also implies that there are advantages to using expressions of the form:

```
keycode number = keysymname
```

This expression syntax requires you to be aware of default keycode/keysym assignments. Also, if you explicitly assign a keysym to a particular keycode, it's much easier to keep track of what you're doing and retrace your steps if necessary. On the down side, though keysyms are portable, keycodes may vary from server to server. Thus, expressions using this syntax cannot be ported to other systems.

14.4.7 Displaying and Changing the Pointer Map

If you want to change the assignment of logical pointer buttons to physical buttons, you should first display the current pointer map with the `-pp` option to *xmodmap*. A typical pointer map appears in Figure 14-8.

```
There are 3 pointer buttons defined.
Physical Button
Button  Code
    1   1
    2   2
    3   3
```

Figure 14-8. Pointer map

This is a fairly simple map: the physical buttons are listed on the left and the corresponding logical functions (button codes) are listed on the right.

These are typical assignments for a right-handed person: the first logical button is the leftmost button, designed to be pressed by the right index finger. The *xmodmap* client allows you to reassign logical buttons so that the pointer can be more easily used with the left hand.

The *xmodmap* client allows you to change the pointer map.* There are two *xmodmap* expressions: one to assign logical pointer buttons (button codes) to physical buttons; and another to restore the default assignments. The syntax of the expressions is:

`pointer=`***x y z***

Sets the first, second, and third physical buttons to the button codes ***x***, ***y***, and ***z***.

`pointer=default`

Sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a code of 2, etc.).

Being able to change the pointer button assignments is very useful if you happen to be left-handed and would like the rightmost physical button to function as the first logical button (i.e., generate button code 1). To configure the pointer for a southpaw:†

```
% xmodmap -e 'pointer = 3 2 1'
```

If you then display the pointer mappings with `xmodmap -pp`, you get this:

```
There are 3 pointer buttons defined.
Physical Button
Button Code
1 3
2 2
3 1
```

You can then push the first logical button (button code 1, which OPEN LOOK uses for SELECT) with the index finger of your left hand; button code 2, ADJUST, with the second finger, and button code 3, MENU, with the third finger.‡

You can return to the default pointer button assignments by entering:

```
% xmodmap -e 'pointer = default'
```

Note that AT&T-OL's version of the Properties Editor lets you assign the pointer buttons more easily, without using *xmodmap*.

† Remember that the `-n` option, which allows you to see what *xmodmap would* do without performing the changes cannot be used with expressions to change the pointer mapping.

‡ Users of OpenWindows may need to take additional action to convince NeWS clients to switch the pointer buttons around.

14.5 xkeycaps - visual keyboard mapping

Maybe Lee can provide this?

PART THREE: Reference Manual Pages

Note: it was indended that the Online Reference Manuals (UNIX "man pages") would be included in the bound version of this volume. However, for the CD-ROM release we have instead chosen to deliever them as individual Man pages in several formats, so that they can be used directly by your UNIX *man* command if you wish.

PART FOUR: Appendices

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX A

The xterm Terminal Emulator

xterm provides you with an X window with a terminal within it. Anything you can do using a standard terminal, you can do in an *xterm* window. Like the *shelltool* terminal emulator described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, this client can be used to create multiple terminal windows, each of which can run any programs available on the underlying operating system. Once you have an *xterm* window on your screen, you can use it to run other clients.

The version of *xterm* shipped with OpenWindows is identical to the MIT X11 xterm. The version shipped with AT&T-OL has been substantially overhauled for OPEN LOOK conformance. To add to the confusion, this "OLlified" version of *xterm* is distributed under the name *olterm* with some System V Release 4 systems, such as Dell Computers' version of X. In this chapter we describe the common xterm version, and describe some aspects of *olterm* by mentioning the differences from *xterm*. We'll use the term *xterm* to describe both versions, and we'll refer to the AT&T-OL version as *olterm*.

You can bring up more than one *xterm* window at a time. For example, you might want to list the contents of a directory in one window while you edit a file in another window. Although you can display output simultaneously in several windows, you can type in only one window at a time.

When you start an *xterm* process on the command line in one *xterm* window, the second *xterm* inherits the environment variables of the first (including the DISPLAY setting); the second shell also starts in the working directory of the first shell.

The basic operation of *xterm* should be obvious to anyone familiar with a terminal. You should be able to work productively immediately.

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

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

Among the less obvious features of *xterm* is a dual functionality. By default, *xterm* emulates a DEC VT102 terminal, a common alphanumeric terminal type. AT&T-OL calls this mode the "AT&T 6386 console" emulation mode. However, *xterm* can also emulate the Tektronix 4014 terminal, an obsolete terminal that was used to display simple computer graphics before bit-mapped window systems became prevalent. For each *xterm* process, you can switch between these two types of terminal windows. You can display both a VT102 and a Tektronix window at the same time but only one of them can be the "active" window, i.e., the window receiving input and output. Hypothetically, you could be editing in the VT102 window while looking at graphics in the Tektronix window.

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

The VT Fonts menu lets you change the font used to display text in the VT102 window. You may want to change the font for a number of reasons. Perhaps you need a larger font to read text more easily; or maybe you want to use a smaller font to reduce the size of a window while a program is running and you don't need to monitor its progress.

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

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

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

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

A.1 Terminal Emulation and the xterm Terminal Type

Anyone who has used a variety of terminals knows that they don't all work the same way. As a terminal emulator, an *xterm* window must be assigned a terminal type that tells the system how the window should operate, that is, what type of terminal it should emulate. When *xterm* is assigned an invalid terminal type, the window does not display properly at all times, particularly when using a text editor such as *vi*. If one of your login files (*.login*, *.profile*, *.cshrc*, etc.) currently specifies a default terminal type, you will need to replace

this with a type valid for *xterm*. (If none of your login files specifies a terminal type, *xterm* automatically searches the file of TERMCAP entries for the first valid entry.)

xterm can emulate a variety of terminal types, which are listed on the client reference page in Part Three of this guide. An *xterm* window most successfully emulates a terminal when it has been assigned the terminal type xterm. For the xterm terminal type to be recognized on your system, the system administrator will have had to add it to the file containing valid TERMCAP entries. (The xterm TERMCAP entry is supplied with the standard release of X.) If this has not been done, the system will not recognize the xterm terminal type. In these cases, try the vt100 terminal type, which also generally works well, or use one of the other types listed on the client reference page.

See , and the *xterm* reference page in Part Three of this guide for information about customizing the *termcap* file.

A.2 Resizing an xterm Window

xterm sets the TERMCAP environment variable for the dimensions of the window you create. Clients (including *xterm*) use this TERMCAP information to determine the physical dimensions of input and output to the window.

If you resize an *xterm* window, programs running within the window must be notified so they can adjust the dimensions of input and output to the window. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from BSD 4.3), *xterm* will use these facilities to notify programs running in the window whenever it is resized. However, if your operating system does not support terminal resizing capabilities, you may need to request explicitly that TERMCAP be updated to reflect the resized window.

The *resize* client sends a special escape sequence to the *xterm* window and *xterm* sends back the current size of the window. The results of *resize* can be redirected to a file that can then be sourced to update TERMCAP. To update TERMCAP to match a window's changed dimensions, enter:

```
% resize > filename
```

and then execute the resulting shell command file:

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

or:

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

TERMCAP will be updated and the dimensions of the text within the window will be adjusted accordingly.

An easier way is to use command substitution. If your regular shell on UNIX is the C shell, you can define this alias for *resize*:

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

On the Korn or Bourne shell, you can use

```
rs() { eval 'resize' }
```

Then use `rs` to update the TERMCAP entry to reflect a window's new dimensions.

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

A.3 Using the Athena Scrollbar

When using *xterm*, you are not limited to viewing the 24 lines displayed in the window at one time. By default, *xterm* actually remembers the last 64 lines that have appeared in the window. If the window has a scrollbar, you can scroll up and down through the saved text. This section describes **only** the MIT and OpenWindows *xterm* scrolling; the OPEN LOOK scrollbar used in the AT&T-OL *xterm* (and in most Sun OpenWindows applications) is described in the section *Using the OPEN LOOK Scrollbar* in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

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

```
% xterm -sb &
```

To display all *xterm* windows with a scrollbar by default, set `scrollBar` in your *.Xresources* file, as described in Chapter 12, *Setting Resources*. The appropriate resource setting is illustrated below:

```
XTerm*scrollBar: true
```

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

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

xterm's scrollbar is created by the Athena Scrollbar widget. (As we'll see in subsequent chapters, several of the standard X clients use Athena scrollbars, while all OPEN LOOK-conforming applications use OPEN LOOK scrollbars.) An Athena scrollbar looks and operates differently than a scrollbar provided by a *(OL application (that is, one created using one of the *(OL toolkits or widget sets), as described in Chapter 5, *The cmdtool/shelltool Terminal Emulator*. If you're accustomed to using an *(OL (or even a Motif or Macintosh) scrollbar, the Athena scrollbar may take some getting used to, buy you will soon find that it does the same thing in a different way. While most other scrollbars have separate parts to invoke different types of scrolling, the Athena scrollbar moves text according to which pointer button you use and how you use it.

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

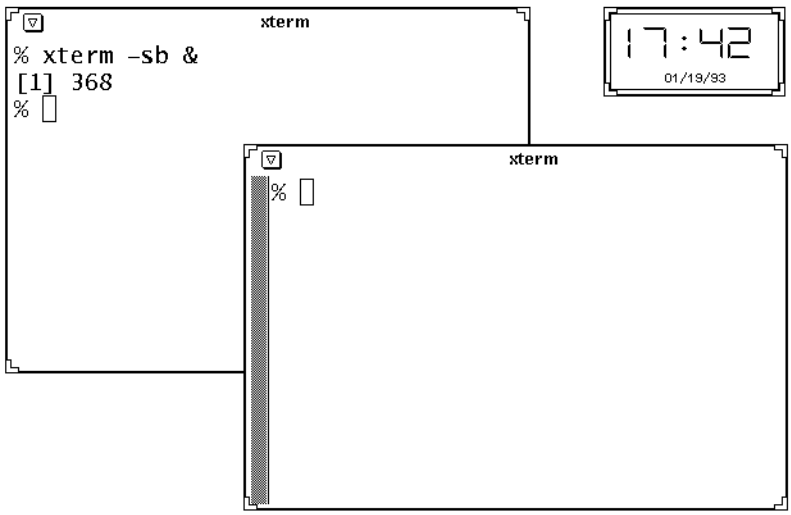


Figure A-1. An xterm window with a scrollbar

The Athena scrollbar has two parts: a *thumb* (the highlighted area within the scrollbar) which moves within the *scroll region*. The thumb displays the position and amount of text currently showing in the window relative to the amount saved. When an *xterm* window with a scrollbar is first created, the thumb fills the entire scrollbar. As more text is saved, the size of the thumb decreases. The number of lines saved is 64 by default but an alternative can be specified with either the `-sl` command line option or the `saveLines` value in a resources file.

You scroll through the saved text using various pointer commands. When the pointer is positioned in the scrollbar, the cursor changes to a two-headed arrow. The scrollbar commands are summarized in Table 5-1.

Table A-1. Athena Scrollbar Commands

To move text in this direction:	Place pointer on scrollbar and:	Notes:
Either up or down	Hold down second pointer button and drag thumb.	Text follows pointer movement.
Down	Click first pointer button.	Scrolls towards latest saved text (towards bottom of window).

Table A-1. Athena Scrollbar Commands

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

The first command in Table 5-1 involves dragging the text in the window using the second pointer button. This command is the simplest and offers the most control over how much scrolling takes place. To drag the text in this manner: first place the pointer on the scrollbar; press and hold down the second pointer button; then drag the thumb up and down. Notice that text moves as you move the thumb. If you drag up, the window scrolls back toward the beginning of information saved in the window. If you drag down, the window scrolls forward toward the end of information in the window. When you release the button, the window displays the text at that location. This makes it easy to get to the top of the data by pressing the second button, dragging the thumb to the top of the scroll region, and releasing the pointer button.

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

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

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

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

A.4 Copying and Pasting Text Selections — MIT and OpenWindows Only

Once your *xterm* window is created, you can select text to copy and paste within the same or other *xterm* windows using the pointer. You don't need to be in a text editor to copy and paste. You can also copy or paste text to and from the command line. As with the Scrollbar, there are two different ways of doing this, the Athena way and the OPEN LOOK way. And

again, the MIT and OpenWindows versions of *xterm* use the Athena way, while the AT&T-OL version of *xterm* (and all OPEN LOOK applications) use the OPEN LOOK version, which is outlined in the section *The Edit Menu* below, and described in more detail in *Copying and Pasting Text Selections* in Chapter 5, *The cmdtool/shelltool Terminal Emulator*.

Text copied into memory using the pointer is saved in a global cut buffer and also becomes what is known as the PRIMARY text "selection."[1]

Both the contents of the cut buffer and the contents of the PRIMARY text selection are globally available to all clients. When you paste text into an *xterm* window, by default the contents of the PRIMARY selection are pasted. If no text is in the PRIMARY selection, the contents of the cut buffer (called CUT_BUFFER0), are pasted. (In most cases, these will be the same.)

Copying and pasting is one way in which clients exchange information, in this case, text. Later in this chapter, we'll consider some of the complications that can arise when copying and pasting between applications that save information differently. For now, however, let's see how to copy and paste text between *xterm* windows.

A.4.1 Selecting Text to Copy

There are several ways to select (copy) text. You can select text by individual words or lines, or you can select a passage of text.

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

There are two methods for selecting a passage of text. First, you can make the selection by dragging the pointer: place the pointer at the beginning of the text you want to select; hold down the first button; move the pointer to the end of the desired text; then release the button. The text is highlighted, copied into the global cut buffer (called CUT_BUFFER0) and also made the PRIMARY selection.

The second way to select a passage is even simpler: mark the beginning of the selection by clicking the first pointer button; then mark the end of the selection by clicking the third pointer button. The text between the marks is highlighted, copied into CUT_BUFFER0, and made the PRIMARY selection.

You can select a single word or line simply by clicking. To select a single word, place the pointer on the word and double-click the first button.[2]

To select a single line, place the pointer on the line and triple-click the first button.

1. The PRIMARY selection and the cut buffer are stored as *properties* of the root window. A property is a piece of information associated with a window (or font) and stored in the server, where it can be accessed by any client. The property mechanism permits "cut" text to be stored and later "pasted" into the windows of other clients. See Chapter 1 and Chapter 10 for more about properties and interclient communication.

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

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

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

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

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

Once you have made a selection with the first button, you can extend that selection with the third button. This example shows how this works:

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

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

Place the pointer on the word *graphics* in the sample sentence and select it by double-clicking the first button.

Then press and hold down the third pointer button. Move the pointer away from the word *graphics* to the left or right. A new selection now extends from the last selection (*graphics*) to the pointer's location and looks something like this (the text that is underlined here will normally be shown in reverse video on your screen):

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

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

or:

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

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

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

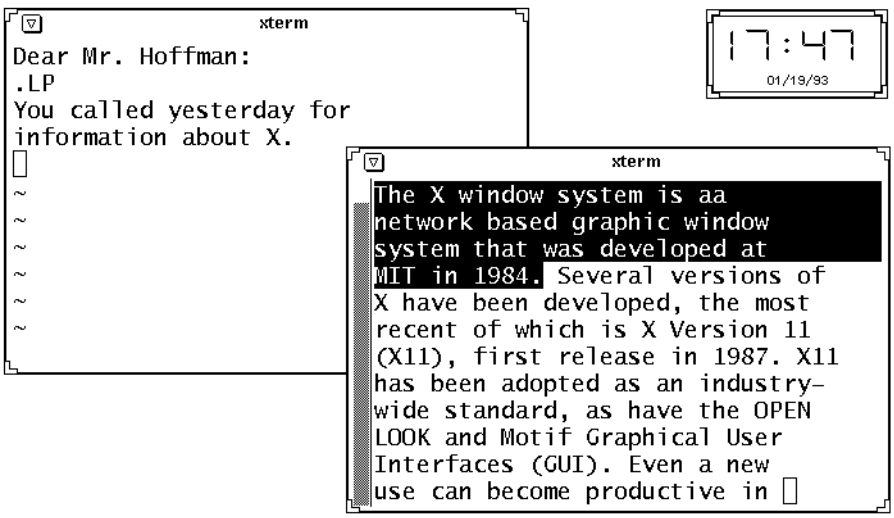


Figure A-2. Highlighted text saved as the PRIMARY selection

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

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

A.4.2 Pasting Text Selections

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

You can paste text either into an open file or at a command line prompt. To paste text into an open file, as illustrated in Figure A-3, click the second button within the window containing the file. The text from the memory area will be inserted at the text editor cursor. (Of course, the file must be in a mode where it is expecting text input, such as the insert mode of an editor.) You can paste the same text as often as you like. The contents of the PRIMARY selection remain until you make another selection.

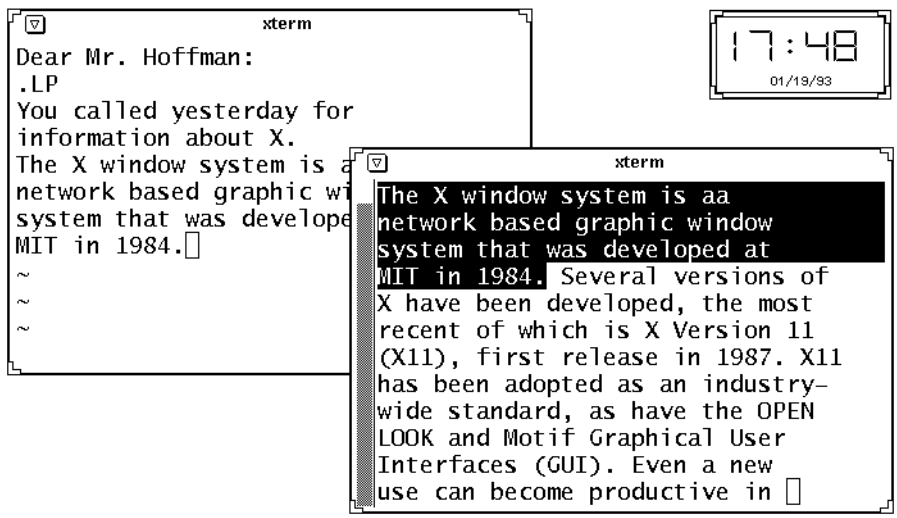


Figure A-3. Pasting text into an open file

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

Note that you can paste text into a window when click-to-type focus is in effect, even if the window does not have the input focus. The act of pasting does not transfer focus either. (Similarly, if you click on a window to focus input, the click is not interpreted as an attempt to start a text selection.)

Keep in mind that you can paste *over* existing text in a file with the *vi* change text commands (such as cw, for change word). For example, you can paste over five words by

specifying the *vi* command 5cw, and then pasting text by clicking the second pointer button. Note that you can paste over existing text in any editor that has an overwrite mode.

A.5 More About Text Selections

Most X11 clients exchange information via selections. The selection mechanism allows data from one client to be copied to another client, and optionally converted to a different format used by the receiving client.

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

Because of the nature of selections (particularly the issue of ownership), the following problems can arise in transferring data:

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

The *xclipboard* client addresses these problems.

Most users will probably not encounter the second problem. You are probably doing all of your copying and pasting between *xterm* windows. If you've made a selection from an *xterm* window and the window is killed, the *selection* contents are lost. However, contents of an older X11 mechanism, the "cut buffer", remain intact and are pasted instead. (Since all *xterm* windows interpret ASCII text, the translation capabilities of the selection mechanism are not needed.)

Problems involving the loss of selections are more likely to happen if you are transferring information between clients that require information to be in different formats. If you are having such problems, you can customize the clients involved to copy information to what is known as the CLIPBOARD selection.

The CLIPBOARD selection is intended to avert problems of selection ownership by providing centralized ownership. Once the CLIPBOARD owns a selection, the selection can be transferred (and translated), even if the client that previously owned the selection goes away.

You can customize a client to send data to the CLIPBOARD selection by using *event translations*, which are discussed in Chapter 10. See the client reference pages in Part Three of this guide for information on the appropriate translations. For more information on selections and translations, see Volume One, *Xlib Programming Manual*.

A.5.1 Saving Multiple Selections: xclipboard

The *xclipboard* client provides a window in which you can paste multiple text selections and from which you can copy text selections to other windows. Similar to the clipboard feature of the Macintosh operating system, the *xclipboard* is basically a storehouse for text

you may want to paste into other windows, perhaps multiple times. The *xclipboard* window is shown in Figure A-4.

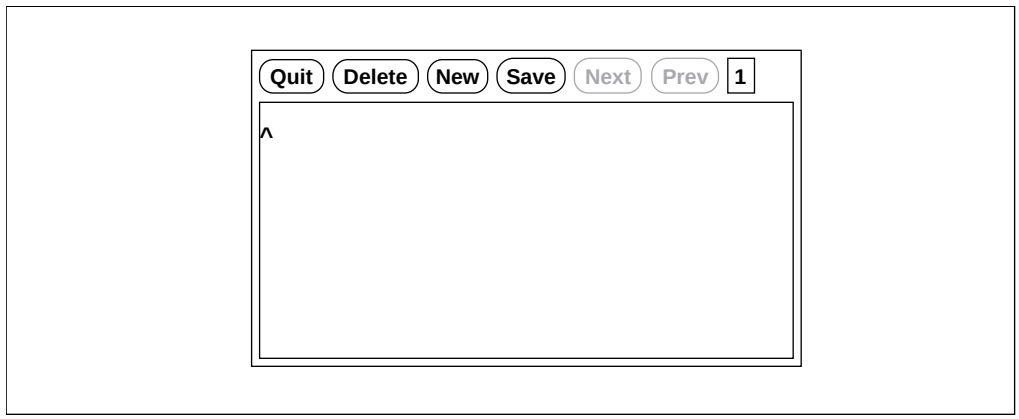


Figure A-4. The xclipboard window

To open an *xclipboard*, type:

```
% xclipboard &
```

You can paste text into the *xclipboard* window using the pointer in the manner described previously and then copy and paste it elsewhere but this is not its intended use. To use the *xclipboard* most effectively, you must do some customization involving a resource file, such as *.Xdefaults*. The necessary steps are described in detail in Chapter 10. For now, suffice it to say that you want to set up the *xclipboard* so that you can select text to be made the CLIPBOARD selection and have that text *automatically pasted* in the *xclipboard* window, as illustrated in XXX.

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

In order to illustrate how the clipboard works, let's presume it has been set up according to the guidelines in Chapter 10. According to those guidelines, you make text the CLIPBOARD selection by selecting it with the first pointer button (as usual) and then, while holding down the first button, clicking the third button. (You could specify another button combination or a button and key combination but we've found this one works pretty well.) The first pointer action makes the text the PRIMARY selection (and it is available to be pasted in another window using the pointer); the second pointer action additionally makes the text the CLIPBOARD selection (and it is automatically sent to the *xclipboard* window).

These guidelines still allow you to select text with the first pointer button alone and that text will be made the PRIMARY selection; however, the text will not automatically be sent to the *xclipboard*. This enables you to make many selections but to direct to the *xclipboard* only those selections you consider important (perhaps those you might want to paste several times).

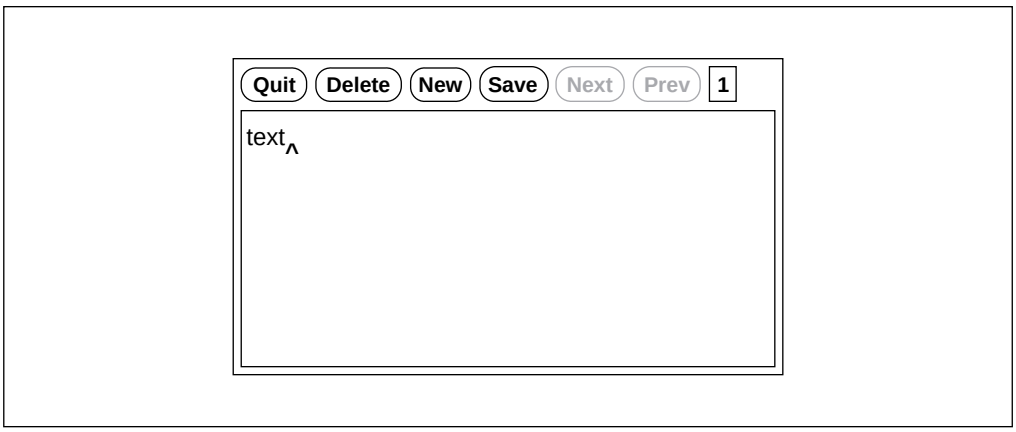


Figure A-5. Selected text appears automatically in the xclipboard window

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

Once you have saved multiple selections, the client's Next and Previous command buttons allow you to move forward and backward among these screens of text. The functionality of the client's command buttons is summarized in Table A-3. They are all selected by clicking the first pointer button.

Table A-3. Command Buttons and Functions

Button	Function
Quit	Causes the application to exit.
Delete	Deletes the current *xclipboard* buffer; the current screenful of text is cleared from the window and the next screenful (or previous, if there is no next) is displayed.
New	Opens a new buffer into which you can insert text; the window is cleared.

Table A-3. Command Buttons and Functions

Button	Function
Next and Previous	Once you have sent multiple selections to the *xclipboard*, Next and Previous allow you to move from one to another (e.g., display them sequentially). Before two or more CLIPBOARD selections are made, these buttons are not available for use. (Their labels will appear in a lighter typeface to indicate this.)

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

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

You select text from the *xclipboard* and paste it where you want it just as you would any text. Just display the text you want in the *xclipboard* window, using Next or **Previous** as necessary. Then select the text using the first pointer button and paste it using the second pointer button.

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

Certain features (and limitations) of the *xclipboard* become apparent only when you make a very large CLIPBOARD selection. Say you select a full *xterm* window of text with the first and third pointer buttons, as described above. The text extends both horizontally and vertically beyond the bounds of a single *xclipboard* screen. (As we suggested earlier, a CLIPBOARD selection can actually span more than one *xclipboard* screen. Pressing Delete will remove all screens that the selection comprises.) When you make a selection that extends beyond the bounds of the *xclipboard* screen (either horizontally, vertically, or both), scrollbars will be activated in the window to allow you to view the entire selection, as shown in Figure A-6.

If the text extends both horizontally and vertically beyond the bounds of the *xclipboard* screen, as it does in Figure A-6 the window will display both horizontal and vertical scrollbars. If the text extends beyond the screen in only one of these two ways, the window will display either a horizontal or vertical scrollbar, as needed.* These scrollbars are selection-specific: they are only displayed as long as the current selection cannot be viewed in its entirety without them. If you move to a previous or subsequent selection that *can* be viewed without scrollbars, the scrollbars will be deactivated.[1]

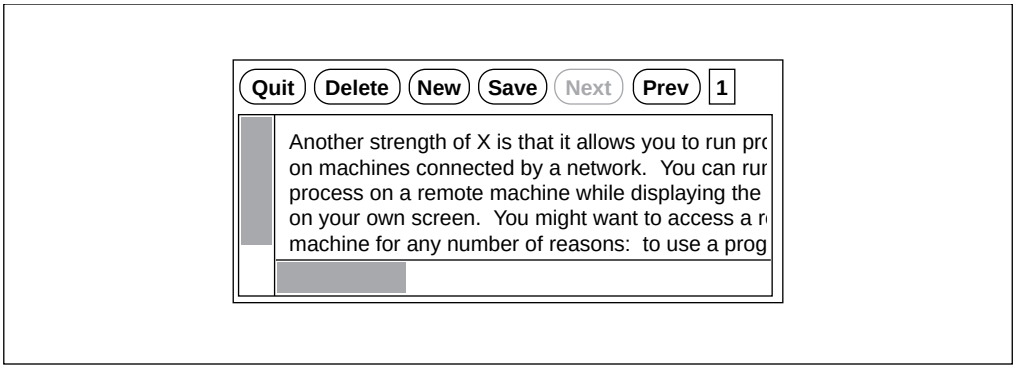


Figure A-6. xclipboard with scrollbars to view large text selection

A.5.1.1 Problems with Large Selections

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

In order to get around this problem and display the actual current selection, press the Previous button. The same long selection (which is, in actuality, the Previous selection) will again be displayed. Then the Next button will be enabled, and you can click on it to display the actual current selection.

A.5.1.2 Editing Text Saved in the xclipboard

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

Be aware that, without performing customization, you can still use *xclipboard* on a very simple level. You can paste text into and copy text from the *xclipboard* window just as you

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

would any other, using the pointer movements described earlier in this chapter. You can also type in the *xclipboard* window and then copy and paste what you've typed. Just move the pointer into the window and try typing. However, keep in mind that this is not the intended use of the *xclipboard*.

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

A.6 Running a Program in a Temporary xterm Window

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

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

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

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

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

A.7 The xterm Menus — MIT, OpenWindows

The MIT X11 Release 5 version of *xterm*, which is included in Sun's OpenWindows Release 3.3, has four different menus, described here. The AT&T-OL version of *xterm* has a different set of menus, and is described in the following section, *The xterm Menus — AT&T-OL*.

- Main Options menu (formerly called **xterm** menu).
- VT Options menu (formerly called **Modes** menu).
- VT Fonts menu (available as of MIT X11 Release 4).
- Tek Options menu (formerly called **Tektronix** menu).

The VT Fonts menu, which allows you to change the *xterm* display font dynamically, was introduced in Release 4. The other three menus are updated versions of menus available in Release 3. As is indicated above, these three menus have been renamed in Release 4. Most

of the *items* available on these menus have not changed in functionality since Release 3, though many have been renamed and some have been reorganized.

As shown in Figure A-7, three of the four *xterm* menus are divided into sections separated by horizontal lines. The top portion of each divided menu contains various modes that can be toggled. (The one exception is the Redraw Window item on the **Main Options** menu, which is a command.) A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state.

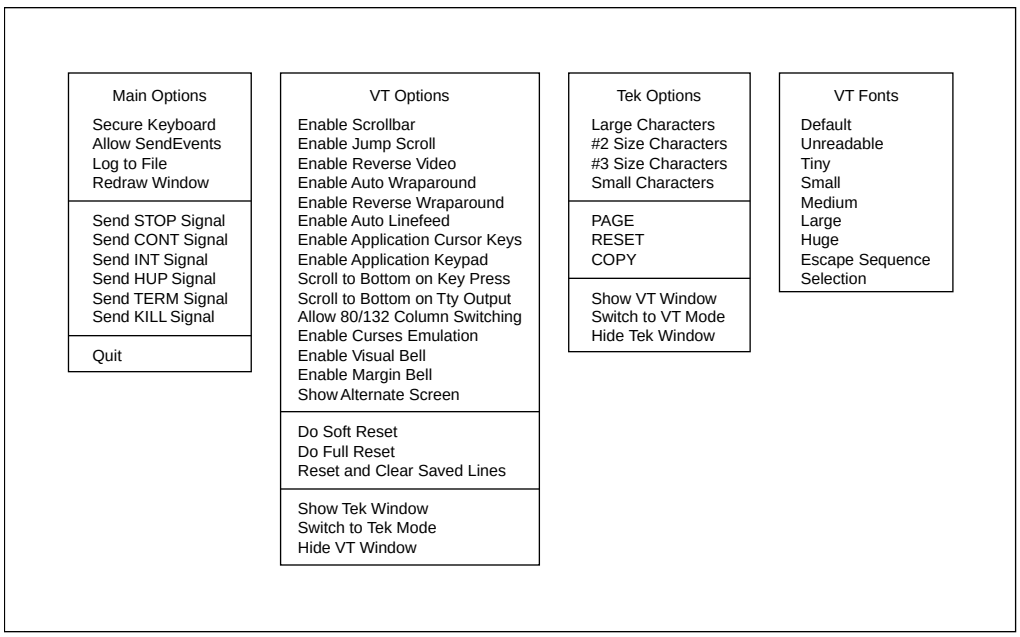


Figure A-7. The X11 Release 5 xterm menus

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

Most mode entries can also be set by command line options when invoking *xterm*, or by entries in a resource startup file (such as *.Xdefaults* or *.Xresources*) as described in Chapter 12, *Setting Resources*. (See the *xterm* reference page in Part Three of this guide for a complete list of command options and resource variables.) The various modes on the menus are very helpful if you've set (or failed to set) a particular mode on the command line and then decide you want the opposite characteristic.

The sections below the modes portion of each menu contain various commands. Selecting one of these commands performs the indicated function. Many of these functions can only be invoked from the *xterm* menus. However, some functions can be invoked in other

ways: for example, from an *mwm* menu, on the command line, by a sequence of keystrokes (such as Control-C). This chapter includes alternatives to some of the menu items which, in certain cases, may be more convenient. Of course, the *xterm* menus can be very helpful when other methods to invoke a function fail.

Menus are displayed by placing the pointer on the window and simultaneously pressing a keyboard key and pointer button. (The exact key and button combinations are described below with each menu.) When you're using a window manager, such as *mwm*, that provides a titlebar or frame, the pointer must rest within the window proper—not on any window decoration. (Note that the pointer be within the window, even if click-to-type focus is enabled. See Chapter 1 for a discussion of focus policy.)

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

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

In this discussions of the four *xterm* menus, we'll consider some of the more useful items as well as some alternatives to menu items. For more complete information about each menu, see the *xterm* reference page in Part Three of this guide.

A.7.1 The Main Options Menu

The Main Options menu, shown in Figure A-8, allows you to set certain modes and to send signals (such as SIGHUP) that affect the *xterm* process.

To bring up the Main Options menu, move the pointer to the *xterm* window you want to change, hold down the Control key, and press the first (usually the left) pointer button.* The pointer changes to the menu pointer, and this menu of three modes and eight commands appears. (You can release the Control key but must continue to press the first pointer button to hold the Main Options menu in the window.)[1]

Note that Main Options menu items apply only to the *xterm* window the pointer is in when you display the menu. To effect changes in another *xterm*, you must move the pointer to that window, display the menu, and specify the items you want.

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

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

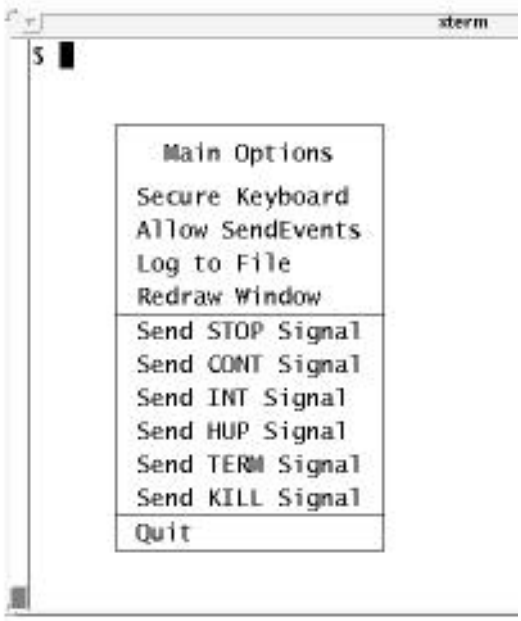


Figure A-8. The xterm main menu

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

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

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

When you enable Secure Keyboard mode, the foreground and background colors of the *xterm* window will be exchanged (as if you had enabled the Reverse Video mode from the

VT Options menu), as shown in Figure A-9. When you disable Secure Keyboard mode, the colors will be switched back.

Figure A-9. Reverse video is enabled when the keyboard is secure

Be aware that only one X client at a time can secure the keyboard. Thus, if you have enabled Secure Keyboard mode in one *xterm*, you will not be allowed to enable it in another *xterm* until you disable it in the first. If Secure Keyboard mode is not available when you request it, the colors will not be switched and a bell will sound. Note also that the window manager cannot obtain keystrokes while an *xterm* is in Secure Keyboard mode, so that keyboard accelerators for window manager functions will not operate; they will probably appear as gibberish in the secure *xterm* window. Finally, note that other X applications' pop-up dialogs will not be able to get the keyboard either.

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

Be aware that Secure Keyboard will be disabled automatically if you iconify the *xterm* window, or start *olwm* or *mwm* or another window manager that provides a titlebar or other window decoration. (You can enable Secure Keyboard mode once the new window man-

ager is running, though.) This limitation is due to the X protocol. When the mode is disabled, the colors will be switched back and the bell will sound to warn you.

Though intended to counteract a security weakness, the Secure Keyboard mode toggle can also be used to get around a weakness in X. As described in Chapter 6, *Using the OPEN LOOK Window Manager*, if the window manager dies, it's possible that the focus can be lost—i.e., the focus is no longer directed to any application window. Selecting Secure Keyboard mode for any *xterm* should cause that window to grab the focus again.

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

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

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

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

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

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

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

A.7.2 VT Options Menu

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

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

To bring up the VT Options menu, move the pointer to the *xterm* window, hold down the Control key, and then press and hold down the second pointer button. (You can release the Control key but must continue to press the second button to keep the VT Options menu in the window.) The menu shown in Figure A-10 appears.

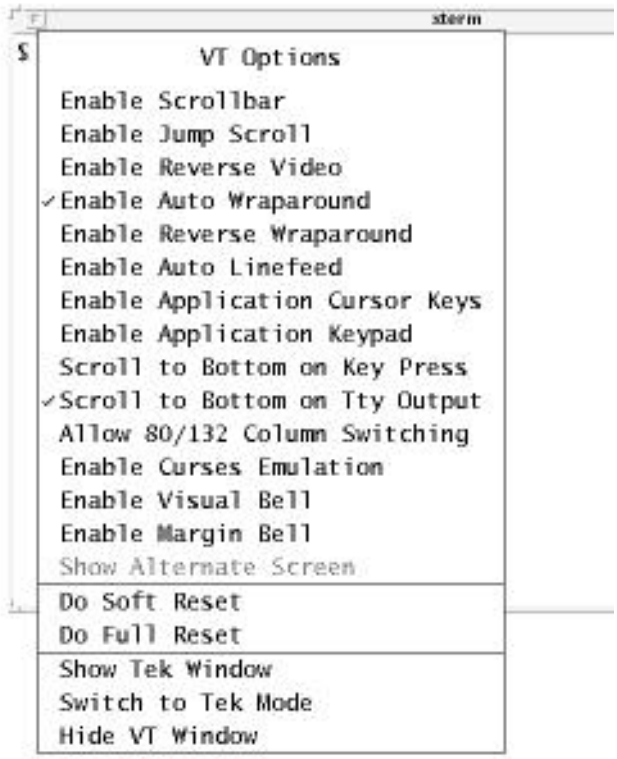


Figure A-10. The VT Options menu

Check marks indicate the active modes. For example, Jump Scroll, **Auto Wraparound**, and **Scroll to Bottom on Tty Output** are active in the VT Options menu displayed in Figure A-10. These are the only modes active by default.[1]

To turn off one of these modes, move the menu pointer to that mode and release the second button.

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

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

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

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

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

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

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

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

A.7.3 VT Fonts Menu

The VT Fonts menu is a welcome Release 4 innovation. It allows you to change the display font of an *xterm* window while the window is running. To bring up the VT Fonts menu, move the pointer inside the *xterm* window. Press and hold down the Control key on the keyboard and press the third (usually the right) pointer button. The VT Fonts menu is shown in Figure A-11.

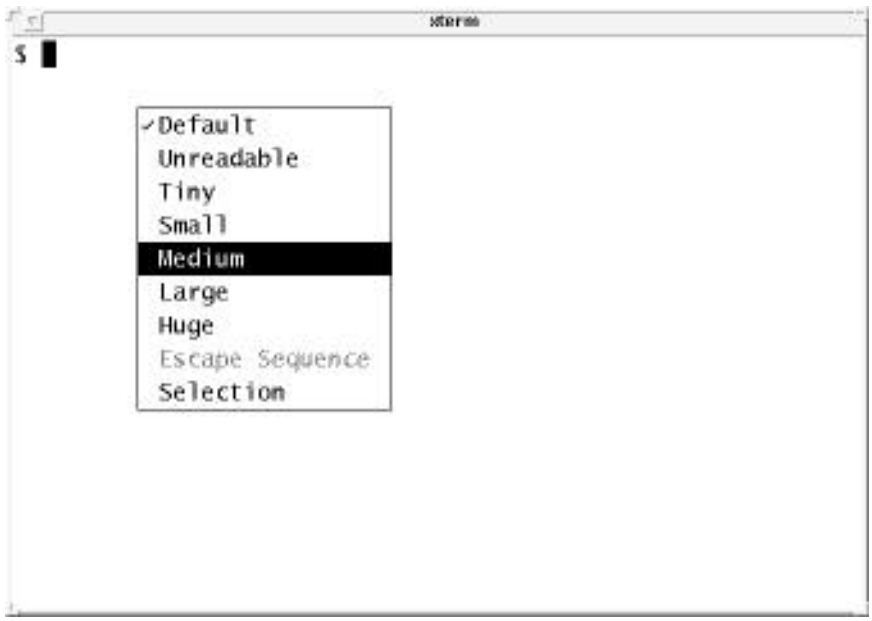


Figure A-11. VT Fonts menu

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

The items Default, **Tiny**, **Small**, Medium, and **Large** can be toggled to set the font displayed in the *xterm* window. The font can be changed any number of times to accommodate a variety of uses. You might choose to use a large font for editing a file (chances are you've chosen a large enough default font, though). You could then change to a smaller font while a process is running since you don't need to be reading or typing in that *xterm*. Changing the font also changes the size of the window.

! *There are also default settings for the Tiny, **Small**, Medium, and **Large** fonts. They are all constant-width fonts from the directory /usr/lib/X11/fonts/misc and are listed in Table A-4.*

Table A-4. VT Fonts Menu Defaults

Menu Item	Default Font
Tiny	nil2
Small	6x10
Medium	8x13
Large	9x15

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

Be aware that you can specify your own Tiny, **Small**, Medium, and **Large** fonts using entries in a resource startup file such as.*Xdefaults* or *.Xresources*. The corresponding resource names are `font1`, `font2`, `font3`, and `font4`. See Chapter 6 for more information about available fonts. See Chapter 10 for instructions on how to set resource variables.

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

A.7.4 Tek Options Menu

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

To display the Tek Options menu, move the pointer inside the Tektronix window. Press and hold down the Control key on the keyboard and press the second pointer button. The Tek Options menu appears.. With this menu you set the size of the text in the Tektronix window and select some commands.

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

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

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

A.8 The xterm Menus—AT&T/olterm Version

The AT&T-OL version of *xterm* has most of the same requests available in menus, but they are organized differently, in an attempt to bring *xterm* into compliance with the OPEN LOOK specification. The main menu, activated by pressing the MENU (usually right) pointer button, looks like this:

```
xterm
Edit ->
Redraw
Soft Reset
Full Reset
Properties...
Show Tek window
Interrupt
Hangup
Terminate
Kill
```

Edit gets you into the edit menu, to be described shortly. *Redraw* redisplays the window, just like *Refresh* on the workspace menu. *Soft Reset*, *Full Reset* and *Show Tek Window* are the same as the corresponding items on the *VT Options* menu in the MIT version. The *Properties...* item starts up a properties window that has most of the items from the first section of the *VT Options* shown in Figure A-10 above. And the last four items send the same signals as do the corresponding items in the *Main Options* menu in the MIT version above.

A.8.1 AT&T-OL xterm Edit Menu

The *Edit* menu looks like this:

```
Send
Paste
Copy
Cut(X)
```

These provide a simple cut and paste facility similar to that described for the MIT *xterm* above. You can select text using the SELECT (left) pointer button. Just click and hold SELECT before the first character you want to select, and drag the pointer across all the text you want to select, and release SELECT. Then you can copy this text into your *xterm* window just by selecting *Send* from the *Edit* menu. The *Copy* operation copies selected text into a clipboard, while *Paste* copies the clipboard into the current window or text field. Think of *Send* as a combination of *Copy* and *Paste*.

The OPEN LOOK text selection and copying mechanism is described in more detail in Chapter 5, *The cmdtool/shelltool Terminal Emulator*, in the section *Copying and Pasting Text Selections.*

A.8.2 AT&T-OL xterm Properties Window

The *Properties* window lets you set most of the settings that are in the first part of the *VT Options* in the MIT and OpenWindows version of *xterm*. The property window, which is pinnable, has these Checkbox (exclusive settings), which have the same meaning as the same-named items described previously for the MIT version of *xterm*:

```
Visual BellLogging
Jump ScrollReverse Video
Auto WraparoundReverse Wraparound
Auto LinefeedApplication Cursor
Application PadScroll Bar
Margin BellSecure Keyboard
Curses Resize
```

A.8.3 AT&T-OL xterm Tek mode menus

The *Tektronix Mode xterm Menu* contains these selections:

```
PAGE
RESET
COPY
Redraw
Properties...
Hide VT window
Interrupt
Hangup
Terminate
Kill
```

The first three, in upper case, are the same as the middle section of the MIT version's *Tek Options* menu. The next two do what you'd expect: *Redraw* redisplays the screen, and *Properties* brings up a Properties window. *Hide VT window* causes the "AT&T 6386" (VT102) window to be unmapped. THe last four have the same meaning as on the main *xterm* menu here, and on the *Main Options* menu in the MIT version.

The *Tek Properties* window lets you choose one of four sizes — Large, Medium, Small, and Tiny — for the Tek display. These are identical to the like-named items in the *VT Fonts* menu of the MIT version.

A.8.4 AT&T-OL Keyboard Shortcuts

All the xterm menus have keyboard shortcuts, shown by underlining in the menu listings above. In addition, the following shortcuts are available. The key bindings assume an AT&T 6386 WGS keyboard.

Table A-5. AT&T-OL Xterm Key Sequences

Function	Default Binding	xterm use
Scroll Up	<Prior>	Scroll text up
Down	<Next>	scroll text down
Up	<Ctrl><Prior>	Scroll up one page
Down	<Ctrl><Next>	Scroll down one page
Top	<Alt><Prior>	Jump scroll to top of text
Bottom	<Alt><Next>	Jump scroll to end of text
Scrollbar menu	Ctrl-r	Activate Scrollbar menu
Menu	Ctrl-m	Activate xterm main menu
Cut	<Shift><Delete>	Cut selected text
Copy	<Ctrl><Insert>	Copy selected text into clipboard
Paste	<Shift><Insert>	Paste in copied text into window

In Tek mode, only Menu (Ctrl-m) works.

1.9 Changing Fonts in xterm Windows

xterm includes a VT Fonts menu that allows you to change fonts on the fly. We discussed most of the menu entries above. These items require a detailed understanding of font naming such as that given in Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*. So we've saved them until the end of this Appendix..

1.9.0.1 The Great Escape

Though it is by no means obvious, *xterm* allows you to change the display font by sending an escape sequence, along with the new font name, to the terminal window. Once you change the font in this way, the **Escape Sequence** item on the *xterm* VT Fonts menu becomes available and choosing it toggles the font you first specified with the escape sequence. (In effect, whatever font you specify using the escape sequence is stored in memory as the menu's **Escape Sequence** font selection.)

You send an escape sequence to the terminal window by using the UNIX *echo* (1) command. The escape sequence to change the *xterm* display font comprises these keystrokes:

```
Esc ] 50 ; fontname Control-G
```

To clarify, these keystrokes are: the Escape key, the right bracket (]), the number 50, a semicolon (;), a ***fontname***, and the Control-G key combination. We've shown the keystrokes with spaces between them for readability, but when you type the sequence on the command line, there should be no spaces. Note also that to supply this sequence as an argument to *echo*, you must enclose it in quotes:

```
% echo "Esc]50;fontnameControl-G"
```

These are the literal keys you type. However, be aware that when you type these keys as specified, the command line will not look exactly like this. Certain keys, such as Escape, and key combinations, such as Control-G, are represented by other symbols on the command line. When you type the previous key sequence, the command line will actually look like this:

```
% echo "^[]50;fontname^G"
```

Pressing the Escape key generates the "^[" symbol; typing the Control-G key combination generates "^G." You can use a full fontname, an alias, or a wildcarded font specification as the font name. You should be aware that if the wildcarded specification matches more than one font, you will get the first font in the search path that matches. For example:

```
% echo "^[]50;*courier*^G"
```

will get you a 10-point courier bold oblique. The advantage of being able to change the display font with an escape sequence is that it allows you to add another font to your choices on the fly.[1]

Changing the fonts associated with the **Tiny**, **Small**, **Medium**, and **Large** menu items is a more laborious process. It involves specifying other fonts in a resource file, making those resources available to the server, and then running another *xterm* process. (See Chapter 12, *Setting Resources*, for more information.) However, you can change the font specified by the **Escape Sequence** menu item as often as you want during the current *xterm* process, simply by typing the escape sequence described previously.

Now that we've looked at the mechanics of the escape sequence, let's consider its practical use. Say you want to run a program in an *xterm* window and you want to be able to read the output easily, but you would like the window to be moderately small. You discover that toggling the **Medium** font, the `8x13` font by default, makes the window a good size, but the typeface is too light to be read easily. (We presume you are using the default menu fonts and have not customized them using a resource file.) You could dynamically change the display font to a bold font of the same size by entering the following command line:

```
% echo "Esc]50;8x13boldControl-G"
```

The *xterm* font becomes the desired `8x13bold`, a good choice; in addition, the **Escape Sequence** item of the **VT Fonts** menu becomes available for selection. This menu item allows you to toggle the `8x13bold` font at any time during the *xterm* process. Thus, you

1. Specifying a font with an escape sequence affects only the current *xterm* window and enables only that window's **Escape Sequence** menu selection.

could switch back to any of the other fonts available on the menu (**Small**, **Large**, etc.) and then use **Escape Sequence** to again select `8x13bold`.

This font will remain the **Escape Sequence** font for the duration of the *xterm* process, unless you again change the display font with an escape sequence. If you enter another font name using the escape sequence described above, the window will display in that new font and the **Escape Sequence** menu item will toggle it.

1.9.0.2 The Selection Menu Item

The **Selection** menu item allows you to toggle a font whose name you've previously "selected." The font name could be selected with the pointer, for example, from *xlsfonts* output, using the "cut-and-paste" techniques described in Appendix A, *The xterm/olterm Terminal Emulator*. It is far more likely, though, that you would use this menu item after selecting a font with *xfontsel*. This menu item was clearly designed with *xfontsel* in mind. (If no text is currently selected, this menu item appears in a lighter typeface, indicating that it is unavailable.)

The main limitation of this menu item is that it uses the *last text selected* as the font name, regardless of what that text is. If you select a font name, that name is only available through **Selection** until you use the pointer to select other text. Since cutting and pasting text is one of the most useful features of *xterm*, you will probably be making frequent selections. If the last selected text was not a valid font name, toggling **Selection** will not change the display font, and a beep will inform you that the toggle failed.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX B

OpenWindows and X11 Standard Fonts

This appendix shows the standard display fonts available in OpenWindows Release 3 and in X11 Release 5. The images contained in this appendix are window dumps created with our own program, called *xshowfonts*, the code for which is included in the CD-ROM.

This appendix does not show how to add fonts to the OpenWindows server or the MIT X server, norr use of the X11R5 "font server"; these are covered in Volume Eight, *X Window System Administrator's Guide*. We do discuss adding Type 1 PostScript fonts to OpenWindows 3.3 and later.

B.1 Pictures of Fonts

This appendix includes pictures of some representative fonts from the standard X distributions. No two distributions of X11 (except two copies of the same release direct from the X Consortium) contain exactly the same set of fonts! Not every font may be supported by particular server vendors, and some vendors may supplement the set.

Section B.2, "Fonts in the X11R5/6 (and modern OpenWindows) Servers," lists the fonts in the X11R5 and modern OpenWindows directories. Section B.3, "Fonts in the xnews Server," lists the fonts in the older OpenWindows servers. The first column lists the name of the file in which the font is stored (without the filenameextension); the second column lists the actual font name. See Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*, for information about font naming conventions.

PICTURES of the different font families supplied in the standard X11 distribution appear in Section B.4, "Font Samples . We show just the fonts in the *75dpi* directory. The *100dpi* directory contains the same fonts stored in the *75dpi* directory but for 100 dots per inch monitors. Keep in mind that all of the fonts in the *75dpi* and *100dpi* directories are available in 8-, 10-, 12-, 14-, 18-, and 24-point sizes. Each page shows fonts of various sizes, weights, and styles. We include the source for *xshowfonts.c*, the program written at O'Reilly & Associates to make these displays, at the end of the appendix.. Section B.5,

"Font Encodings," shows you, using *xfd*, one example of each of the unique *encodings*, or character sets, available.[1]

All of the characters in each font are shown more-or-less actual size. But since no two monitiros have exactly the same pixel density, these fonts would appear in a different size on your monitor.

B.2 Fonts in the X11R5/6 (and modern OpenWindows) Servers

On the X Consortium X11R5 and R6 releases, as well as OpenWindows after Release 3.2, the standard fonts are stored in several directories, as shown in Table B-1

Table B-1. MIT X11 Font Directories

Directory (relaitve to /usr/lib/X11 or /usr/openwin/home/lib/X11)	Contents
.../fonts/misc	Fxed-width fonts, the cursor font, other miscellaneous fonts.
.../fonts/75dpi	Fixed- and variable-width fonts, 75 dots per inch.
...//fonts/100dpi	Fixed- and variable-width fonts, 100 dpi.
.../fonts/Speedo	Charter and Courier outline fonts from Bitstream
.../fonts/F3 (and F3bitmaps)	Sun "F3" fonts (Sun only)
.../fonts/Type1	Display PostScript Type 1 fonts (Sun and other DPS vendors only)

The standard server can accept fonts in any of several formats:

Table 2-2. X11R5 Font File Formats

Suffix	File Format
.afm	Adobe ASCII format metric file
.bdf	Adobe BDF 2.1 bitmap file

1. If you want to use this program yourself, you probably don't want to type the sourcein, so it's in the src/utils dir4ectory of the CD-ROM. Or, you can obtain the source from uunet.uu.net via anonymous *ftp* or *uucp*. See the Preface for more information.

Table 2-2. X11R5 Font File Formats

Suffix	File Format
.snf	Server Normal Form
.spd	Speedo Scaleable form
.ff	Font Family file
.fm	NeWS font metric file
.ps	PostScript font file (usually Type Three)
.vft	Vfont (Berkeley) bitmap font

Here is a listing of the fonts.

Table B-3. Fonts in the misc Directory

Filename	Font name
6x12.pcf.Z	-misc-fixed-medium-r-semicondensed--12-110-75-75-c-60-iso8859-1
6x13.pcf.Z	-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-1
6x10.pcf.Z	-misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1
7x13.pcf.Z	-misc-fixed-medium-r-normal--13-120-75-75-c-70-iso8859-1
7x14.pcf.Z	-misc-fixed-medium-r-normal--14-130-75-75-c-70-iso8859-1
clR8x12.pcf.Z	-schumacher-clean-medium-r-normal--12-120-75-75-c-80-iso8859-1
6x9.pcf.Z	-misc-fixed-medium-r-normal--9-90-75-75-c-60-iso8859-1
clR8x13.pcf.Z	-schumacher-clean-medium-r-normal--13-130-75-75-c-80-iso8859-1
clR8x10.pcf.Z	-schumacher-clean-medium-r-normal--10-100-75-75-c-80-iso8859-1
5x7.pcf.Z	-misc-fixed-medium-r-normal--7-70-75-75-c-50-iso8859-1
clR8x16.pcf.Z	-schumacher-clean-medium-r-normal--16-160-75-75-c-80-iso8859-1
clR8x14.pcf.Z	-schumacher-clean-medium-r-normal--14-140-75-75-c-80-iso8859-1
clR8x8.pcf.Z	-schumacher-clean-medium-r-normal--8-80-75-75-c-80-iso8859-1
5x8.pcf.Z	-misc-fixed-medium-r-normal--8-80-75-75-c-50-iso8859-1
clR9x15.pcf.Z	-schumacher-clean-medium-r-normal--15-150-75-75-c-90-iso8859-1
clR6x8.pcf.Z	-schumacher-clean-medium-r-normal--8-80-75-75-c-60-iso8859-1
clR5x6.pcf.Z	-schumacher-clean-medium-r-normal--6-60-75-75-c-50-iso8859-1
clR7x8.pcf.Z	-schumacher-clean-medium-r-normal--8-80-75-75-c-70-iso8859-1
clR4x6.pcf.Z	-schumacher-clean-medium-r-normal--6-60-75-75-c-40-iso8859-1
clR5x8.pcf.Z	-schumacher-clean-medium-r-normal--8-80-75-75-c-50-iso8859-1
clR6x6.pcf.Z	-schumacher-clean-medium-r-normal--6-60-75-75-c-60-iso8859-1

Table B-3. Fonts in the misc Directory

Filename	Font name
6x13B.pcf.Z	-misc-fixed-bold-r-semicondensed--13-120-75-75-c-60-iso8859-1
12x24rk.pcf.Z	-sony-fixed-medium-r-normal--24-170-100-100-c-120-jisx0201.1976-0
7x13B.pcf.Z	-misc-fixed-bold-r-normal--13-120-75-75-c-70-iso8859-1
7x14B.pcf.Z	-misc-fixed-bold-r-normal--14-130-75-75-c-70-iso8859-1
clR6x12.pcf.Z	-schumacher-clean-medium-r-normal--12-120-75-75-c-60-iso8859-1
clR6x13.pcf.Z	-schumacher-clean-medium-r-normal--13-130-75-75-c-60-iso8859-1
clR6x10.pcf.Z	-schumacher-clean-medium-r-normal--10-100-75-75-c-60-iso8859-1
clR7x12.pcf.Z	-schumacher-clean-medium-r-normal--12-120-75-75-c-70-iso8859-1
clR7x10.pcf.Z	-schumacher-clean-medium-r-normal--10-100-75-75-c-70-iso8859-1
clR7x14.pcf.Z	-schumacher-clean-medium-r-normal--14-140-75-75-c-70-iso8859-1
8x13.pcf.Z	-misc-fixed-medium-r-normal--13-120-75-75-c-80-iso8859-1
8x16.pcf.Z	-sony-fixed-medium-r-normal--16-120-100-100-c-80-iso8859-1
clR5x10.pcf.Z	-schumacher-clean-medium-r-normal--10-100-75-75-c-50-iso8859-1
9x15.pcf.Z	-misc-fixed-medium-r-normal--15-140-75-75-c-90-iso8859-1
heb6x13.pcf.Z	-misc-fixed-medium-r-semicondensed--13-120-75-75-c-60-iso8859-8
clB8x8.pcf.Z	-schumacher-clean-bold-r-normal--8-80-75-75-c-80-iso8859-1
8x13B.pcf.Z	-misc-fixed-bold-r-normal--13-120-75-75-c-80-iso8859-1
7x14rk.pcf.Z	-misc-fixed-medium-r-normal--14-130-75-75-c-70-jisx0201.1976-0
9x15B.pcf.Z	-misc-fixed-bold-r-normal--15-140-75-75-c-90-iso8859-1
clI8x8.pcf.Z	-schumacher-clean-medium-i-normal--8-80-75-75-c-80-iso8859-1
heb8x13.pcf.Z	-misc-fixed-medium-r-normal--13-120-75-75-c-80-iso8859-8
decsess.pcf.Z	decw$session
clB8x12.pcf.Z	-schumacher-clean-bold-r-normal--12-120-75-75-c-80-iso8859-1
clB8x13.pcf.Z	-schumacher-clean-bold-r-normal--13-130-75-75-c-80-iso8859-1
clB8x10.pcf.Z	-schumacher-clean-bold-r-normal--10-100-75-75-c-80-iso8859-1
clB8x16.pcf.Z	-schumacher-clean-bold-r-normal--16-160-75-75-c-80-iso8859-1
clB8x14.pcf.Z	-schumacher-clean-bold-r-normal--14-140-75-75-c-80-iso8859-1
clB9x15.pcf.Z	-schumacher-clean-bold-r-normal--15-150-75-75-c-90-iso8859-1
olcursor.pcf.Z	-sun-open look cursor-----12-120-75-75-p-455-sunolcursor-1
Cmr-Bold14.pcf.Z	-sun-cmr-bold-r-normal--14-140-72-72-m-100-sun-fontspecific
hanglg16.pcf.Z	-daewoo-gothic-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0
8x16rk.pcf.Z	-sony-fixed-medium-r-normal--16-120-100-100-c-80-jisx0201.1976-0
clB6x12.pcf.Z	-schumacher-clean-bold-r-normal--12-120-75-75-c-60-iso8859-1
clB6x10.pcf.Z	-schumacher-clean-bold-r-normal--10-100-75-75-c-60-iso8859-1
hanglm16.pcf.Z	-daewoo-mincho-medium-r-normal--16-120-100-100-c-160-ksc5601.1987-0

Table B-3. Fonts in the misc Directory

Filename	Font name
jiskan24.pcf.Z	-jis-fixed-medium-r-normal--24-230-75-75-c-240-jisx0208.1983-0
hanglm24.pcf.Z	-daewoo-mincho-medium-r-normal--24-170-100-100-c-240-ksc5601.1987-0
jiskan16.pcf.Z	-jis-fixed-medium-r-normal--16-150-75-75-c-160-jisx0208.1983-0
cursor.pcf.Z	cursor
Serif12.pcf.Z	-sun-serif-medium-r-normal-serif-12-120-72-72-m-70-iso8859-1
Serif10.pcf.Z	-sun-serif-medium-r-normal-serif-10-100-72-72-m-70-iso8859-1
Serif11.pcf.Z	-sun-serif-medium-r-normal-serif-11-110-72-72-m-70-iso8859-1
Serif16.pcf.Z	-sun-serif-medium-r-normal-serif-16-160-72-72-m-90-iso8859-1
Serif14.pcf.Z	-sun-serif-medium-r-normal-serif-14-140-72-72-m-80-iso8859-1
deccurs.pcf.Z	decw$cursor
clI6x12.pcf.Z	-schumacher-clean-medium-i-normal--12-120-75-75-c-60-iso8859-1
Screen6.pcf.Z	-sun-screen-medium-r-normal--6-60-72-72-m-40-sun-fontspecific
Screen7.pcf.Z	-sun-screen-medium-r-normal--7-70-72-72-m-60-sun-fontspecific
Gallant19.pcf.Z	-sun-gallant-demi-r-normal--19-190-72-72-m-120-iso8859-1
olgl19.pcf.Z	-sun-open look glyph-----19-190-75-75-p-163-sunolglyph-1
olgl12.pcf.Z	-sun-open look glyph-----12-120-75-75-p-116-sunolglyph-1
olgl24.pcf.Z	-sun-open look glyph-----24-240-75-75-p-206-sunolglyph-1
k14.pcf.Z	-misc-fixed-medium-r-normal--14-130-75-75-c-140-jisx0208.1983-0
olgl10.pcf.Z	-sun-open look glyph-----10-100-75-75-p-106-sunolglyph-1
olgl16.pcf.Z	-sun-open look glyph-----16-160-75-75-p-137-sunolglyph-1
olgl20.pcf.Z	-sun-open look glyph-----20-200-75-75-p-172-sunolglyph-1
olgl14.pcf.Z	-sun-open look glyph-----14-140-75-75-p-136-sunolglyph-1
nil2.pcf.Z	-misc-nil-medium-r-normal--2-20-75-75-c-10-misc-fontspecific
Cmr14.pcf.Z	-sun-cmr-medium-r-normal--14-140-72-72-m-90-sun-fontspecific
10x20.pcf.Z	-misc-fixed-medium-r-normal--20-200-75-75-c-100-iso8859-1
12x24.pcf.Z	-sony-fixed-medium-r-normal--24-170-100-100-c-120-iso8859-1
Screen-Bold12.pcf.Z	-sun-screen-bold-r-normal--12-120-72-72-m-80-iso8859-1
Screen-Bold16.pcf.Z	-sun-screen-bold-r-normal--16-160-72-72-m-100-iso8859-1
Screen-Bold14.pcf.Z	-sun-screen-bold-r-normal--14-140-72-72-m-90-iso8859-1
Screen12.pcf.Z	-sun-screen-medium-r-normal--12-120-72-72-m-70-iso8859-1
Screen11.pcf.Z	-sun-screen-medium-r-normal--11-110-72-72-m-70-iso8859-1
Screen16.pcf.Z	-sun-screen-medium-r-normal--16-160-72-72-m-90-iso8859-1
Screen14.pcf.Z	-sun-screen-medium-r-normal--14-140-72-72-m-80-iso8859-1

Here are the fonts in the 75dpi directory:

Table B-4. Fonts in the 75dpi Directory

Filename	Font names
courO08.pcf.Z	-adobe-courier-medium-o-normal--8-80-75-75-m-50-iso8859-1
courO18.pcf.Z	-adobe-courier-medium-o-normal--18-180-75-75-m-110-iso8859-1
courO14.pcf.Z	-adobe-courier-medium-o-normal--14-140-75-75-m-90-iso8859-1
courO12.pcf.Z	-adobe-courier-medium-o-normal--12-120-75-75-m-70-iso8859-1
courO24.pcf.Z	-adobe-courier-medium-o-normal--24-240-75-75-m-150-iso8859-1
courO10.pcf.Z	-adobe-courier-medium-o-normal--10-100-75-75-m-60-iso8859-1
lubB08.pcf.Z	-b&h-lucidabright-demibold-r-normal--8-80-75-75-p-47-iso8859-1
lubB18.pcf.Z	-b&h-lucidabright-demibold-r-normal--18-180-75-75-p-107-iso8859-1
lubB19.pcf.Z	-b&h-lucidabright-demibold-r-normal--19-190-75-75-p-114-iso8859-1
lubB12.pcf.Z	-b&h-lucidabright-demibold-r-normal--12-120-75-75-p-71-iso8859-1
lubB24.pcf.Z	-b&h-lucidabright-demibold-r-normal--24-240-75-75-p-143-iso8859-1
lubB10.pcf.Z	-b&h-lucidabright-demibold-r-normal--10-100-75-75-p-59-iso8859-1
lubB14.pcf.Z	-b&h-lucidabright-demibold-r-normal--14-140-75-75-p-84-iso8859-1
courB08.pcf.Z	-adobe-courier-bold-r-normal--8-80-75-75-m-50-iso8859-1
courB18.pcf.Z	-adobe-courier-bold-r-normal--18-180-75-75-m-110-iso8859-1
lubI14.pcf.Z	-b&h-lucidabright-medium-i-normal--14-140-75-75-p-80-iso8859-1
courBO08.pcf.Z	-adobe-courier-bold-o-normal--8-80-75-75-m-50-iso8859-1
lubI12.pcf.Z	-b&h-lucidabright-medium-i-normal--12-120-75-75-p-67-iso8859-1
lubI24.pcf.Z	-b&h-lucidabright-medium-i-normal--24-240-75-75-p-136-iso8859-1
courBO18.pcf.Z	-adobe-courier-bold-o-normal--18-180-75-75-m-110-iso8859-1
lubI10.pcf.Z	-b&h-lucidabright-medium-i-normal--10-100-75-75-p-57-iso8859-1
termB14.pcf.Z	-dec-terminal-bold-r-normal--14-140-75-75-c-80-iso8859-1
courB12.pcf.Z	-adobe-courier-bold-r-normal--12-120-75-75-m-70-iso8859-1
courB24.pcf.Z	-adobe-courier-bold-r-normal--24-240-75-75-m-150-iso8859-1
courB10.pcf.Z	-adobe-courier-bold-r-normal--10-100-75-75-m-60-iso8859-1
courBO14.pcf.Z	-adobe-courier-bold-o-normal--14-140-75-75-m-90-iso8859-1
courBO12.pcf.Z	-adobe-courier-bold-o-normal--12-120-75-75-m-70-iso8859-1
courBO24.pcf.Z	-adobe-courier-bold-o-normal--24-240-75-75-m-150-iso8859-1
lubI08.pcf.Z	-b&h-lucidabright-medium-i-normal--8-80-75-75-p-45-iso8859-1
courB14.pcf.Z	-adobe-courier-bold-r-normal--14-140-75-75-m-90-iso8859-1
courBO10.pcf.Z	-adobe-courier-bold-o-normal--10-100-75-75-m-60-iso8859-1
lubI18.pcf.Z	-b&h-lucidabright-medium-i-normal--18-180-75-75-p-102-iso8859-1
lubI19.pcf.Z	-b&h-lucidabright-medium-i-normal--19-190-75-75-p-109-iso8859-1
lubR08.pcf.Z	-b&h-lucidabright-medium-r-normal--8-80-75-75-p-45-iso8859-1

Table B-4. Fonts in the 75dpi Directory

Filename	Font names
lubR18.pcf.Z	-b&h-lucidabright-medium-r-normal--18-180-75-75-p-103-iso8859-1
lubR19.pcf.Z	-b&h-lucidabright-medium-r-normal--19-190-75-75-p-109-iso8859-1
helvBO08.pcf.Z	-adobe-helvetica-bold-o-normal--8-80-75-75-p-50-iso8859-1
helvBO18.pcf.Z	-adobe-helvetica-bold-o-normal--18-180-75-75-p-104-iso8859-1
lubR12.pcf.Z	-b&h-lucidabright-medium-r-normal--12-120-75-75-p-68-iso8859-1
lubR24.pcf.Z	-b&h-lucidabright-medium-r-normal--24-240-75-75-p-137-iso8859-1
helvBO14.pcf.Z	-adobe-helvetica-bold-o-normal--14-140-75-75-p-82-iso8859-1
lubR10.pcf.Z	-b&h-lucidabright-medium-r-normal--10-100-75-75-p-56-iso8859-1
helvBO12.pcf.Z	-adobe-helvetica-bold-o-normal--12-120-75-75-p-69-iso8859-1
helvBO24.pcf.Z	-adobe-helvetica-bold-o-normal--24-240-75-75-p-138-iso8859-1
helvBO10.pcf.Z	-adobe-helvetica-bold-o-normal--10-100-75-75-p-60-iso8859-1
lubR14.pcf.Z	-b&h-lucidabright-medium-r-normal--14-140-75-75-p-80-iso8859-1
lubBI14.pcf.Z	-b&h-lucidabright-demibold-i-normal--14-140-75-75-p-84-iso8859-1
lubBI12.pcf.Z	-b&h-lucidabright-demibold-i-normal--12-120-75-75-p-72-iso8859-1
lubBI24.pcf.Z	-b&h-lucidabright-demibold-i-normal--24-240-75-75-p-143-iso8859-1
lubBI10.pcf.Z	-b&h-lucidabright-demibold-i-normal--10-100-75-75-p-59-iso8859-1
lubBI08.pcf.Z	-b&h-lucidabright-demibold-i-normal--8-80-75-75-p-48-iso8859-1
lubBI18.pcf.Z	-b&h-lucidabright-demibold-i-normal--18-180-75-75-p-107-iso8859-1
lubBI19.pcf.Z	-b&h-lucidabright-demibold-i-normal--19-190-75-75-p-114-iso8859-1
courR08.pcf.Z	-adobe-courier-medium-r-normal--8-80-75-75-m-50-iso8859-1
courR18.pcf.Z	-adobe-courier-medium-r-normal--18-180-75-75-m-110-iso8859-1
courR12.pcf.Z	-adobe-courier-medium-r-normal--12-120-75-75-m-70-iso8859-1
courR24.pcf.Z	-adobe-courier-medium-r-normal--24-240-75-75-m-150-iso8859-1
courR10.pcf.Z	-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1
courR14.pcf.Z	-adobe-courier-medium-r-normal--14-140-75-75-m-90-iso8859-1
charB08.pcf.Z	-bitstream-charter-bold-r-normal--8-80-75-75-p-50-iso8859-1
charB18.pcf.Z	-bitstream-charter-bold-r-normal--19-180-75-75-p-119-iso8859-1
charB12.pcf.Z	-bitstream-charter-bold-r-normal--12-120-75-75-p-75-iso8859-1
charB24.pcf.Z	-bitstream-charter-bold-r-normal--25-240-75-75-p-157-iso8859-1
charB10.pcf.Z	-bitstream-charter-bold-r-normal--10-100-75-75-p-63-iso8859-1
charB14.pcf.Z	-bitstream-charter-bold-r-normal--15-140-75-75-p-94-iso8859-1
charI14.pcf.Z	-bitstream-charter-medium-i-normal--15-140-75-75-p-82-iso8859-1
charI12.pcf.Z	-bitstream-charter-medium-i-normal--12-120-75-75-p-65-iso8859-1
charI24.pcf.Z	-bitstream-charter-medium-i-normal--25-240-75-75-p-136-iso8859-1
charI10.pcf.Z	-bitstream-charter-medium-i-normal--10-100-75-75-p-55-iso8859-1

Table B-4. Fonts in the 75dpi Directory

Filename	Font names
charI08.pcf.Z	-bitstream-charter-medium-i-normal--8-80-75-75-p-44-iso8859-1
charI18.pcf.Z	-bitstream-charter-medium-i-normal--19-180-75-75-p-103-iso8859-1
charR08.pcf.Z	-bitstream-charter-medium-r-normal--8-80-75-75-p-45-iso8859-1
charR18.pcf.Z	-bitstream-charter-medium-r-normal--19-180-75-75-p-106-iso8859-1
luRS08.pcf.Z	-b&h-lucida-medium-r-normal-sans-8-80-75-75-p-45-iso8859-1
luRS18.pcf.Z	-b&h-lucida-medium-r-normal-sans-18-180-75-75-p-106-iso8859-1
luRS19.pcf.Z	-b&h-lucida-medium-r-normal-sans-19-190-75-75-p-108-iso8859-1
charR12.pcf.Z	-bitstream-charter-medium-r-normal--12-120-75-75-p-67-iso8859-1
charR24.pcf.Z	-bitstream-charter-medium-r-normal--25-240-75-75-p-139-iso8859-1
charR10.pcf.Z	-bitstream-charter-medium-r-normal--10-100-75-75-p-56-iso8859-1
luRS14.pcf.Z	-b&h-lucida-medium-r-normal-sans-14-140-75-75-p-81-iso8859-1
luRS12.pcf.Z	-b&h-lucida-medium-r-normal-sans-12-120-75-75-p-71-iso8859-1
luRS24.pcf.Z	-b&h-lucida-medium-r-normal-sans-24-240-75-75-p-136-iso8859-1
charR14.pcf.Z	-bitstream-charter-medium-r-normal--15-140-75-75-p-84-iso8859-1
luRS10.pcf.Z	-b&h-lucida-medium-r-normal-sans-10-100-75-75-p-58-iso8859-1
lutRS08.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-8-80-75-75-m-50-iso8859-1
lutRS18.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-18-180-75-75-m-110-iso8859-1
lutRS19.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-19-190-75-75-m-110-iso8859-1
lutRS14.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-14-140-75-75-m-90-iso8859-1
lutRS12.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-12-120-75-75-m-70-iso8859-1
lutRS24.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-24-240-75-75-m-140-iso8859-1
lutRS10.pcf.Z	-b&h-lucidatypewriter-medium-r-normal-sans-10-100-75-75-m-60-iso8859-1
luIS14.pcf.Z	-b&h-lucida-medium-i-normal-sans-14-140-75-75-p-82-iso8859-1
luIS12.pcf.Z	-b&h-lucida-medium-i-normal-sans-12-120-75-75-p-71-iso8859-1
luIS24.pcf.Z	-b&h-lucida-medium-i-normal-sans-24-240-75-75-p-136-iso8859-1
luIS10.pcf.Z	-b&h-lucida-medium-i-normal-sans-10-100-75-75-p-59-iso8859-1
luIS08.pcf.Z	-b&h-lucida-medium-i-normal-sans-8-80-75-75-p-45-iso8859-1
luIS18.pcf.Z	-b&h-lucida-medium-i-normal-sans-18-180-75-75-p-105-iso8859-1
techB14.pcf.Z	-dec-terminal-bold-r-normal--14-140-75-75-c-80-dec-dectech
luIS19.pcf.Z	-b&h-lucida-medium-i-normal-sans-19-190-75-75-p-108-iso8859-1
timBI14.pcf.Z	-adobe-times-bold-i-normal--14-140-75-75-p-77-iso8859-1
timBI12.pcf.Z	-adobe-times-bold-i-normal--12-120-75-75-p-68-iso8859-1
timBI24.pcf.Z	-adobe-times-bold-i-normal--24-240-75-75-p-128-iso8859-1
timBI10.pcf.Z	-adobe-times-bold-i-normal--10-100-75-75-p-57-iso8859-1
timBI08.pcf.Z	-adobe-times-bold-i-normal--8-80-75-75-p-47-iso8859-1

Table B-4. Fonts in the 75dpi Directory

Filename	Font names
timBI18.pcf.Z	-adobe-times-bold-i-normal--18-180-75-75-p-98-iso8859-1
luBS08.pcf.Z	-b&h-lucida-bold-r-normal-sans-8-80-75-75-p-50-iso8859-1
luBS18.pcf.Z	-b&h-lucida-bold-r-normal-sans-18-180-75-75-p-120-iso8859-1
luBS19.pcf.Z	-b&h-lucida-bold-r-normal-sans-19-190-75-75-p-122-iso8859-1
luBS14.pcf.Z	-b&h-lucida-bold-r-normal-sans-14-140-75-75-p-92-iso8859-1
luBS12.pcf.Z	-b&h-lucida-bold-r-normal-sans-12-120-75-75-p-79-iso8859-1
luBS24.pcf.Z	-b&h-lucida-bold-r-normal-sans-24-240-75-75-p-152-iso8859-1
luBS10.pcf.Z	-b&h-lucida-bold-r-normal-sans-10-100-75-75-p-66-iso8859-1
lutBS08.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-8-80-75-75-m-50-iso8859-1
lutBS18.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-18-180-75-75-m-110-iso8859-1
lutBS19.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-19-190-75-75-m-110-iso8859-1
lutBS14.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-14-140-75-75-m-90-iso8859-1
lutBS12.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-12-120-75-75-m-70-iso8859-1
lutBS24.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-24-240-75-75-m-140-iso8859-1
lutBS10.pcf.Z	-b&h-lucidatypewriter-bold-r-normal-sans-10-100-75-75-m-60-iso8859-1
timI08.pcf.Z	-adobe-times-medium-i-normal--8-80-75-75-p-42-iso8859-1
timI18.pcf.Z	-adobe-times-medium-i-normal--18-180-75-75-p-94-iso8859-1
timI14.pcf.Z	-adobe-times-medium-i-normal--14-140-75-75-p-73-iso8859-1
timI12.pcf.Z	-adobe-times-medium-i-normal--12-120-75-75-p-63-iso8859-1
timI24.pcf.Z	-adobe-times-medium-i-normal--24-240-75-75-p-125-iso8859-1
timI10.pcf.Z	-adobe-times-medium-i-normal--10-100-75-75-p-52-iso8859-1
timB12.pcf.Z	-adobe-times-bold-r-normal--12-120-75-75-p-67-iso8859-1
timB24.pcf.Z	-adobe-times-bold-r-normal--24-240-75-75-p-132-iso8859-1
timB10.pcf.Z	-adobe-times-bold-r-normal--10-100-75-75-p-57-iso8859-1
timB14.pcf.Z	-adobe-times-bold-r-normal--14-140-75-75-p-77-iso8859-1
timB08.pcf.Z	-adobe-times-bold-r-normal--8-80-75-75-p-47-iso8859-1
timB18.pcf.Z	-adobe-times-bold-r-normal--18-180-75-75-p-99-iso8859-1
tech14.pcf.Z	-dec-terminal-medium-r-normal--14-140-75-75-c-80-dec-dectech
timR12.pcf.Z	-adobe-times-medium-r-normal--12-120-75-75-p-64-iso8859-1
timR24.pcf.Z	-adobe-times-medium-r-normal--24-240-75-75-p-124-iso8859-1
timR10.pcf.Z	-adobe-times-medium-r-normal--10-100-75-75-p-54-iso8859-1
timR14.pcf.Z	-adobe-times-medium-r-normal--14-140-75-75-p-74-iso8859-1
timR08.pcf.Z	-adobe-times-medium-r-normal--8-80-75-75-p-44-iso8859-1
timR18.pcf.Z	-adobe-times-medium-r-normal--18-180-75-75-p-94-iso8859-1
helvB08.pcf.Z	-adobe-helvetica-bold-r-normal--8-80-75-75-p-50-iso8859-1

Table B-4. Fonts in the 75dpi Directory

Filename	Font names
helvB18.pcf.Z	-adobe-helvetica-bold-r-normal--18-180-75-75-p-103-iso8859-1
helvB12.pcf.Z	-adobe-helvetica-bold-r-normal--12-120-75-75-p-70-iso8859-1
helvB24.pcf.Z	-adobe-helvetica-bold-r-normal--24-240-75-75-p-138-iso8859-1
helvB10.pcf.Z	-adobe-helvetica-bold-r-normal--10-100-75-75-p-60-iso8859-1
helvB14.pcf.Z	-adobe-helvetica-bold-r-normal--14-140-75-75-p-82-iso8859-1
term14.pcf.Z	-dec-terminal-medium-r-normal--14-140-75-75-c-80-iso8859-1
helvO08.pcf.Z	-adobe-helvetica-medium-o-normal--8-80-75-75-p-47-iso8859-1
helvO18.pcf.Z	-adobe-helvetica-medium-o-normal--18-180-75-75-p-98-iso8859-1
helvO14.pcf.Z	-adobe-helvetica-medium-o-normal--14-140-75-75-p-78-iso8859-1
helvO12.pcf.Z	-adobe-helvetica-medium-o-normal--12-120-75-75-p-67-iso8859-1
helvO24.pcf.Z	-adobe-helvetica-medium-o-normal--24-240-75-75-p-130-iso8859-1
helvO10.pcf.Z	-adobe-helvetica-medium-o-normal--10-100-75-75-p-57-iso8859-1
helvR08.pcf.Z	-adobe-helvetica-medium-r-normal--8-80-75-75-p-46-iso8859-1
helvR18.pcf.Z	-adobe-helvetica-medium-r-normal--18-180-75-75-p-98-iso8859-1
helvR12.pcf.Z	-adobe-helvetica-medium-r-normal--12-120-75-75-p-67-iso8859-1
helvR24.pcf.Z	-adobe-helvetica-medium-r-normal--24-240-75-75-p-130-iso8859-1
helvR10.pcf.Z	-adobe-helvetica-medium-r-normal--10-100-75-75-p-56-iso8859-1
helvR14.pcf.Z	-adobe-helvetica-medium-r-normal--14-140-75-75-p-77-iso8859-1
luBIS14.pcf.Z	-b&h-lucida-bold-i-normal-sans-14-140-75-75-p-92-iso8859-1
luBIS12.pcf.Z	-b&h-lucida-bold-i-normal-sans-12-120-75-75-p-79-iso8859-1
luBIS24.pcf.Z	-b&h-lucida-bold-i-normal-sans-24-240-75-75-p-151-iso8859-1
luBIS10.pcf.Z	-b&h-lucida-bold-i-normal-sans-10-100-75-75-p-67-iso8859-1
luBIS08.pcf.Z	-b&h-lucida-bold-i-normal-sans-8-80-75-75-p-49-iso8859-1
luBIS18.pcf.Z	-b&h-lucida-bold-i-normal-sans-18-180-75-75-p-119-iso8859-1
luBIS19.pcf.Z	-b&h-lucida-bold-i-normal-sans-19-190-75-75-p-122-iso8859-1
ncenBI14.pcf.Z	-adobe-new century schoolbook-bold-i-normal--14-140-75-75-p-88-iso8859-1
ncenBI12.pcf.Z	-adobe-new century schoolbook-bold-i-normal--12-120-75-75-p-76-iso8859-1
ncenBI24.pcf.Z	-adobe-new century schoolbook-bold-i-normal--24-240-75-75-p-148-iso8859-1
ncenBI10.pcf.Z	-adobe-new century schoolbook-bold-i-normal--10-100-75-75-p-66-iso8859-1
ncenBI08.pcf.Z	-adobe-new century schoolbook-bold-i-normal--8-80-75-75-p-56-iso8859-1
ncenBI18.pcf.Z	-adobe-new century schoolbook-bold-i-normal--18-180-75-75-p-111-iso8859-1
symb12.pcf.Z	-adobe-symbol-medium-r-normal--12-120-75-75-p-74-adobe-fontspecific
symb24.pcf.Z	-adobe-symbol-medium-r-normal--24-240-75-75-p-142-adobe-fontspecific
symb10.pcf.Z	-adobe-symbol-medium-r-normal--10-100-75-75-p-61-adobe-fontspecific
symb14.pcf.Z	-adobe-symbol-medium-r-normal--14-140-75-75-p-85-adobe-fontspecific

Table B-4. Fonts in the 75dpi Directory

Filename	Font names
symb08.pcf.Z	-adobe-symbol-medium-r-normal--8-80-75-75-p-51-adobe-fontspecific
symb18.pcf.Z	-adobe-symbol-medium-r-normal--18-180-75-75-p-107-adobe-fontspecific
ncenB08.pcf.Z	-adobe-new century schoolbook-bold-r-normal--8-80-75-75-p-56-iso8859-1
ncenB18.pcf.Z	-adobe-new century schoolbook-bold-r-normal--18-180-75-75-p-113-iso8859-1
ncenB12.pcf.Z	-adobe-new century schoolbook-bold-r-normal--12-120-75-75-p-77-iso8859-1
ncenB24.pcf.Z	-adobe-new century schoolbook-bold-r-normal--24-240-75-75-p-149-iso8859-1
ncenB10.pcf.Z	-adobe-new century schoolbook-bold-r-normal--10-100-75-75-p-66-iso8859-1
ncenB14.pcf.Z	-adobe-new century schoolbook-bold-r-normal--14-140-75-75-p-87-iso8859-1
ncenI14.pcf.Z	-adobe-new century schoolbook-medium-i-normal--14-140-75-75-p-81-iso8859-1
ncenI12.pcf.Z	-adobe-new century schoolbook-medium-i-normal--12-120-75-75-p-70-iso8859-1
ncenI24.pcf.Z	-adobe-new century schoolbook-medium-i-normal--24-240-75-75-p-136-iso8859-1
ncenI10.pcf.Z	-adobe-new century schoolbook-medium-i-normal--10-100-75-75-p-60-iso8859-1
ncenI08.pcf.Z	-adobe-new century schoolbook-medium-i-normal--8-80-75-75-p-50-iso8859-1
ncenI18.pcf.Z	-adobe-new century schoolbook-medium-i-normal--18-180-75-75-p-104-iso8859-1
ncenR08.pcf.Z	-adobe-new century schoolbook-medium-r-normal--8-80-75-75-p-50-iso8859-1
ncenR18.pcf.Z	-adobe-new century schoolbook-medium-r-normal--18-180-75-75-p-103-iso8859-1
ncenR12.pcf.Z	-adobe-new century schoolbook-medium-r-normal--12-120-75-75-p-70-iso8859-1
ncenR24.pcf.Z	-adobe-new century schoolbook-medium-r-normal--24-240-75-75-p-137-iso8859-1
ncenR10.pcf.Z	-adobe-new century schoolbook-medium-r-normal--10-100-75-75-p-60-iso8859-1
ncenR14.pcf.Z	-adobe-new century schoolbook-medium-r-normal--14-140-75-75-p-82-iso8859-1
charBI14.pcf.Z	-bitstream-charter-bold-i-normal--15-140-75-75-p-93-iso8859-1
charBI12.pcf.Z	-bitstream-charter-bold-i-normal--12-120-75-75-p-74-iso8859-1
charBI24.pcf.Z	-bitstream-charter-bold-i-normal--25-240-75-75-p-154-iso8859-1
charBI10.pcf.Z	-bitstream-charter-bold-i-normal--10-100-75-75-p-62-iso8859-1
charBI08.pcf.Z	-bitstream-charter-bold-i-normal--8-80-75-75-p-50-iso8859-1
charBI18.pcf.Z	-bitstream-charter-bold-i-normal--19-180-75-75-p-117-iso8859-1

The fonts in the 100 dpi fonti difectory are a subset of the above, and it would not be useful

to list them here. The following is the contents of the Speedo directory:

Table B-5. Fonts in the Speedo Directory

Filename	Font names
font0648.spd	-bitstream-charter-medium-r-normal--0-0-0-0-p-0-iso8859-1
font0649.spd	-bitstream-charter-medium-i-normal--0-0-0-0-p-0-iso8859-1
font0709.spd	-bitstream-charter-bold-r-normal--0-0-0-0-p-0-iso8859-1
font0710.spd	-bitstream-charter-bold-i-normal--0-0-0-0-p-0-iso8859-1
font0419.spd	-bitstream-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
font0582.spd	-bitstream-courier-medium-i-normal--0-0-0-0-m-0-iso8859-1
font0583.spd	-bitstream-courier-bold-r-normal--0-0-0-0-m-0-iso8859-1
font0611.spd	-bitstream-courier-bold-i-normal--0-0-0-0-m-0-iso8859-1

The following lists the files in the F3 directory (Sun only). The F3bitmaps directory contains pre-scaled versions of the same fonts, so we do not list them separately.:

Table B-6. Fonts in the F3 Directory (Sun only)

Filename	Font name
NewCenturySchlbk-Bold.f3b	-linotype-new century schoolbook-bold-r-normal--0-0-0-0-p-0-iso8859-1
LucidaSans-BoldItalic.f3b	-b&h-lucida sans-bold-i-normal-sans-0-0-0-0-p-0-iso8859-1
NewCenturySchlbk-BoldItalic.f3b	-linotype-new century schoolbook-bold-i-normal--0-0-0-0-p-0-iso8859-1
Palatino-BoldItalic.f3b	-linotype-palatino-bold-i-normal--0-0-0-0-p-0-iso8859-1
Times-BoldItalic.f3b	-linotype-times-bold-i-normal--0-0-0-0-p-0-iso8859-1
Bembo-BoldItalic.f3b	-monotype-bembo-bold-i-normal--0-0-0-0-p-0-iso8859-1
Bookman-LightItalic.f3b	-urw-itc bookman-light-i-normal--0-0-0-0-p-0-iso8859-1
GillSans-BoldItalic.f3b	-monotype-gill sans-bold-i-normal-sans-0-0-0-0-p-0-iso8859-1
Lucida-BrightDemiBold-Italic.f3b	-b&h-lucida bright-demibold-i-normal--0-0-0-0-p-0-iso8859-1
Lucida-BrightItalic.f3b	-b&h-lucida bright-medium-i-normal--0-0-0-0-p-0-iso8859-1
Rockwell-BoldItalic.f3b	-monotype-rockwell-bold-i-normal--0-0-0-0-p-0-iso8859-1
NewCenturySchlbk-Italic.f3b	-linotype-new century schoolbook-medium-i-normal--0-0-0-0-p-0-iso8859-1
Palatino-Italic.f3b	-linotype-palatino-medium-i-normal--0-0-0-0-p-0-iso8859-1
Bembo-Italic.f3b	-monotype-bembo-medium-i-normal--0-0-0-0-p-0-iso8859-1

Table B-6. Fonts in the F3 Directory (Sun only)

Filename	Font name
ZapfChancery-MediumItalic.f3b	-urw-itc zapfchancery-medium-i-normal--0-0-0-0-p-0-iso8859-1
Rockwell-Bold.f3b	-monotype-rockwell-bold-r-normal--0-0-0-0-p-0-iso8859-1
AvantGarde-DemiOblique.f3b	-urw-itc avant garde-demi-o-normal-sans-0-0-0-0-p-0-iso8859-1
Helvetica-Narrow-Oblique.f3b	-linotype-helvetica-medium-o-narrow-sans-0-0-0-0-p-0-iso8859-1
Lucida-Bright.f3b	-b&h-lucida bright-medium-r-normal--0-0-0-0-p-0-iso8859-1
Courier.f3b	--courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
NewCenturySchlbk-Roman.f3b	-linotype-new century schoolbook-medium-r-normal--0-0-0-0-p-0-iso8859-1
Palatino-Bold.f3b	-linotype-palatino-bold-r-normal--0-0-0-0-p-0-iso8859-1
GillSans.f3b	-monotype-gill sans-medium-r-normal-sans-0-0-0-0-p-0-iso8859-1
Times-Bold.f3b	-linotype-times-bold-r-normal--0-0-0-0-p-0-iso8859-1
LucidaSans-Bold.f3b	-b&h-lucida sans-bold-r-normal-sans-0-0-0-0-p-0-iso8859-1
Times-Italic.f3b	-linotype-times-medium-i-normal--0-0-0-0-p-0-iso8859-1
Bookman-DemiItalic.f3b	-urw-itc bookman-demi-i-normal--0-0-0-0-p-0-iso8859-1
LucidaSansTypewriter-Bold.f3b	-b&h-lucida sans typewriter-bold-r-normal-sans-0-0-0-0-m-0-iso8859-1
Courier-Oblique.f3b	--courier-medium-o-normal--0-0-0-0-m-0-iso8859-1
Helvetica-Narrow-Bold.f3b	-linotype-helvetica-bold-r-narrow-sans-0-0-0-0-p-0-iso8859-1
Bembo.f3b	-monotype-bembo-medium-r-normal--0-0-0-0-p-0-iso8859-1
Courier-Bold.f3b	--courier-bold-r-normal--0-0-0-0-m-0-iso8859-1
AvantGarde-Book.f3b	-urw-itc avant garde-medium-r-normal-sans-0-0-0-0-p-0-iso8859-1
AvantGarde-Demi.f3b	-urw-itc avant garde-demi-r-normal-sans-0-0-0-0-p-0-iso8859-1
Rockwell.f3b	-monotype-rockwell-medium-r-normal--0-0-0-0-p-0-iso8859-1
LucidaSans-Italic.f3b	-b&h-lucida sans-medium-i-normal-sans-0-0-0-0-p-0-iso8859-1
GillSans-Italic.f3b	-monotype-gill sans-medium-i-normal-sans-0-0-0-0-p-0-iso8859-1
Helvetica-Oblique.f3b	-linotype-helvetica-medium-o-normal-sans-0-0-0-0-p-0-iso8859-1
Bookman-Light.f3b	-urw-itc bookman-light-r-normal--0-0-0-0-p-0-iso8859-1
Palatino-Roman.f3b	-linotype-palatino-medium-r-normal--0-0-0-0-p-0-iso8859-1
Bembo-Bold.f3b	-monotype-bembo-bold-r-normal--0-0-0-0-p-0-iso8859-1
Rockwell-Italic.f3b	-monotype-rockwell-medium-i-normal--0-0-0-0-p-0-iso8859-1
Courier-BoldOblique.f3b	--courier-bold-o-normal--0-0-0-0-m-0-iso8859-1

Table B-6. Fonts in the F3 Directory (Sun only)

Filename	Font name
Helvetica-BoldOblique.f3b	-linotype-helvetica-bold-o-normal-sans-0-0-0-0-p-0-iso8859-1
Helvetica-Narrow-BoldOblique.f3b	-linotype-helvetica-bold-o-narrow-sans-0-0-0-0-p-0-iso8859-1
Lucida-BrightDemiBold.f3b	-b&h-lucida bright-demibold-r-normal--0-0-0-0-p-0-iso8859-1
Helvetica-Bold.f3b	-linotype-helvetica-bold-r-normal-sans-0-0-0-0-p-0-iso8859-1
LucidaSansType-writer.f3b	-b&h-lucida sans typewriter-medium-r-normal-sans-0-0-0-0-m-0-iso8859-1
Times-Roman.f3b	-linotype-times-medium-r-normal--0-0-0-0-p-0-iso8859-1
GillSans-Bold.f3b	-monotype-gill sans-bold-r-normal-sans-0-0-0-0-p-0-iso8859-1
Symbol.f3b	--symbol-medium-r-normal--0-0-0-0-p-0-sun-fontspecific
Helvetica.f3b	-linotype-helvetica-medium-r-normal-sans-0-0-0-0-p-0-iso8859-1
AvantGarde-BookOblique.f3b	-urw-itc avant garde-medium-o-normal-sans-0-0-0-0-p-0-iso8859-1
Bookman-Demi.f3b	-urw-itc bookman-demi-r-normal--0-0-0-0-p-0-iso8859-1
Helvetica-Narrow.f3b	-linotype-helvetica-medium-r-narrow-sans-0-0-0-0-p-0-iso8859-1
LucidaSans.f3b	-b&h-lucida sans-medium-r-normal-sans-0-0-0-0-p-0-iso8859-1
ZapfDingbats.f3b	-urw-itc zapfdingbats-medium-r-normal--0-0-0-0-p-0-sun-fontspecific

Most modern commercial versions of X, including OpenWindows, include the Display PostScript option. Fonts for this are normally stored as PostScript Type 1 Ascii (with the extention .pfa). Here are the fonts in the Type1 directory for OpenWindows 3.3:

Table B-7. Fonts in the Type1 (Display PostScript) Directory

Filename	Font name
UTRG____.pfa	-adobe-utopia-medium-r-normal--0-0-0-0-p-0-iso8859-1
UTI_____.pfa	-adobe-utopia-medium-i-normal--0-0-0-0-p-0-iso8859-1
UTB_____.pfa	-adobe-utopia-bold-r-normal--0-0-0-0-p-0-iso8859-1
UTBI____.pfa	-adobe-utopia-bold-i-normal--0-0-0-0-p-0-iso8859-1
cour.pfa	-adobe-courier-medium-r-normal--0-0-0-0-m-0-iso8859-1
couri.pfa	-adobe-courier-medium-i-normal--0-0-0-0-m-0-iso8859-1
courb.pfa	-adobe-courier-bold-r-normal--0-0-0-0-m-0-iso8859-1
courbi.pfa	-adobe-courier-bold-i-normal--0-0-0-0-m-0-iso8859-1

If you wish to add your own Type 1 fonts, you need to use *mkpsres* in addition to setting up the font directory. There are some commercial tools which help you in this process. If you use Adobe software such as Adobe Illustrator, you should use their TypeInstaller pro-

gram, which lets you install fonts from CD-ROM, browse fonts (see Figure 2-1), download fonts to the printer, and get a list of fonts resident in the printer.

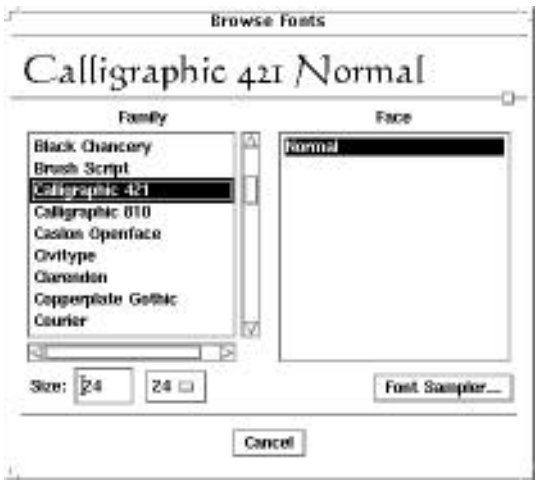


Figure 2-1. Adobe TypeInstaller's Browse Fonts Screen

B.3 Fonts in the xnews Server

On OpenWindows up to Release 3.2, the X server was based on Sun's own server., The standard fonts are stored in the directories shown in Table B-1.

Table B-8. OpenWindows Release 3 Font Directories

Directory	Contents
/usr/openwin/lib/fonts	Font masters (.f3b) and families (.ff) for all scalable fonts.
/usr/openwin/lib/fonts/afm	Adobe Font Metrics for other sofrtware that needs them..
/usr/openwin/lib/fonts/100dpi	Pre-computed bitmaps for standard fonts, 100 dpi.

2.3.1 Font Formats

The OpenWindows server accepts fonts in several formats. The most common are bitmaps, Sun Folio fonts, and Postscript fonts. Since the OpenWindows server is not derived from the MIT server, it does not accept bitmap fonts in the MIT server's internal "SNF" (server normal form) format. Instead, the *convertfont* program is used to convert bitmap files into a form that the server can accept. This is analogous to the use of *bdftosnf* in the MIT server,

and is perfectly in keeping with the X standards, since *convertfont* will accept fonts in the X Consortium standard Bitmap Distribution Format (".bdf" files). However, unlike *bdftosnf*, OpenWindows's *convertfont* can also convert from many other formats into the form needed by the OpenWindows server. Here is a summary of the font file name suffixes and their types:

Table 2-9. OpenWindows Font File Formats

Suffix	File Format	Convert font input?	Output ?
.afb	Adobe ASCII format bitmap font file	y	
.afm	Adobe ASCII format metric file	n/a	
.bdf	MIT (Adobe) BDF 2.1 bitmap file	y	
.f3b	Folio Font format file	y	n
.fb	NeWS binary file	y	
.ff	Font Family file		
.fm	NeWS font metric file	y	
.ps	PostScript font file (usually Type Three)	n	
.vft	Vfont (Berkeley) bitmap font	y	y

The Font Family files are made by *bldfamily*, a program used in conjunction with *convertfont*.

As you can see, *convertfont* can read and write most of the important bitmap font file formats. Here is an example of using it on a font file in the MIT BDF format:.

```
% convertfont kbb.bdf
kbb.bdf->./KlBB2424.fb
% ls -l kbb* K*
-rw-r--r-- 1 ian 69632 Jun 23 15:56 KlBB2424.fb
-rw-rw-r-- 1 ian 133938 Feb 4 11:24 kbb.bdf
%
```

Having generated a "font binary" (.fb) file, you must install it in the appropriate font directory, as described in the following section, and run *bldfamily*. See the reference pages for these two programs in Part Three of this guide for details.

2.3.2 PostScript fonts and ldf

You can also use PostScript Type Three fonts directly in the OpenWindows server. However, installing them in the font directory does not automatically make them available. Presumably because they are so much more memory-expensive than bitmap and Folio fonts, you must explicitly load them. You could load them by hand using *psh*, but it is more convenient to use a special script called *ldf* (load font). Invoked with no arguments, it lists the available PostScript (.ps) fonts in all the directories in $FONTPATH. For example:

```
% ldf
ldf: loads PostScript defined NeWS fonts.
Usage: ldf fontname
Where fontname may be one of the following:
    fontnamePostScript name
    -----------------------
     Dijkstra  Dijkstra
     HrshCyr  Hershey-Cyrillic
     HrshGoth  Hershey-Gothic
     HrshGrk  Hershey-Greek
     HrshGrkN  Hershey-Greek-Narrow
     HrshGrkp  Hershey-Greekp
     HrshI  Hershey-Italic
     HrshIN  Hershey-Italic-Narrow
     HrshIW  Hershey-Italic-Wide
     HrshRom  Hershey-Roman
     HrshRomN  Hershey-Roman-Narrow
     HrshRomW  Hershey-Roman-Wide
     HrshRomd  Hershey-Romand
     HrshRomp  Hershey-Romanp
     HrshScr  Hershey-Script
     HrshScrN  Hershey-Script-Narrow
     HaeberliWriting
```

The first column is the filename (relative to one of the font directories), and the second, if different,[1] is the name you use to refer to the font. The Hershey fonts are the ubiquitous public-domain Hershey fonts, converted to an outline format for use here. HaeberliWriting is the handwriting font shown in the *Pageview* example in Chapter 9, *Graphics Clients*. It appeared in the USENET newsgroups comp.graphics, comp.fonts, comp.lang.postscript, and several system-specific groups courtesy of Paul Haeberli of Silicon Graphics, <paul@manray.asd.sgi.com>, on 10 July 1990. Silicon Graphics was the first major workstation vendor outside of Sun to offer NeWS support on their workstations. Paul writes: "Here's another outline font in Adobe type 3 format. This is a replica of my handwriting, drawn onto the screen of an IRIS, and auto traced into a PostScript Font."

We've installed it on our reference system in the directory */usr/local/lib/fonts*, which is mentioned in $FONTPATH. So to use it, all I need to is give the command

1. *Ldf* is a shell script; it locates the PostScript name by *grep*ping ' ^/FontName', so if the font doesn't have its definition of FontName beginning in column one of the line it's on, ldf prints it as blank!

```
ldf HaeberliWriting
```

(in fact this command is in my *.xinitrc* file), and I can use it in any NeWS clients. For example, in the "text demo" mentioned earlier, I can select it from the font menu, and get this:

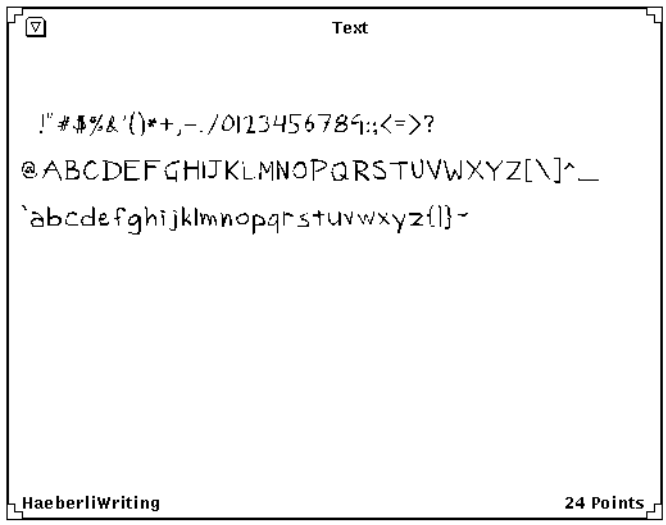


Figure 2-2. Text Demo Program showing HaeberliWriting

And, as shown in Chapter 9, *Graphics Clients*, I can use it in normal PostScript. However, these Type 3 fonts do not have X11 names, so they do not show up in an *xlsfonts* listing, and cannot be used by X11 clients:

```
% xfd -fn 'HaeberliWriting-12'
xfd: error: Unable to open font HaeberliWriting-12!
%
```

However, they are a lot of fun if you know how to make use of them. There are many freely-available Type3 fonts available on the Internet and on BBS systems..

B.4 Font Samples

B.5 Font Encodings

A font encoding is a representation of the characters in a font and their locations within a font. This is to enable applications to know that the characters they need are available on

each font that has a given encoding. The common font encodings used with X11 are described in Table B-10, and are pictured on the following pages.

Table B-10. Font Encodings used with X11

Name	Description
iso8859-1	Extended ASCII character set
iso8859-8	Same with additional characters
adobe-fontspecific	For use by Adobe fonts
dec-dectech	For use by DEC fonts
jisx0201.1976	Japanese character set
jisx0208.1983	Japanese character set, revised
ksc5601.1987	Korean/Asicn character set
sun-fontspeci	For use by Sun special-purpose fonts
sun-open look cursor	Cursors for use with Sun OPEN LOOK software
sun-open look glyph	Picture elemeents used by Sun OPEN LOOK software

The following figures show xfd displays of the standard font encodings (just the first page of wide character sets is shown). The shell script which produced these bitmaps, *show-encodings*, is included in the *src* directory on the CD-ROM..

B

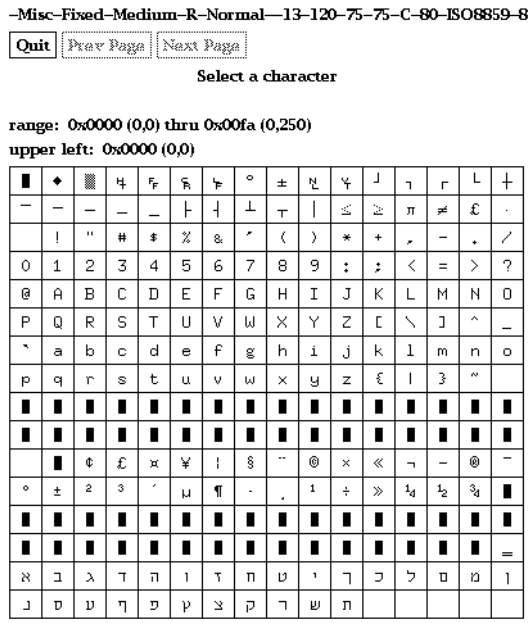


Figure 2-3. ISO8859-1 Encoding

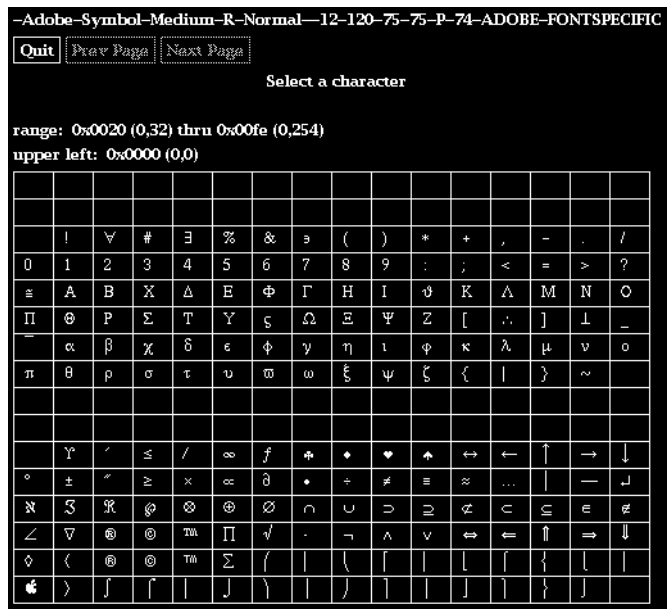


Figure 2-4. Adobe-Fontspecific Encoding

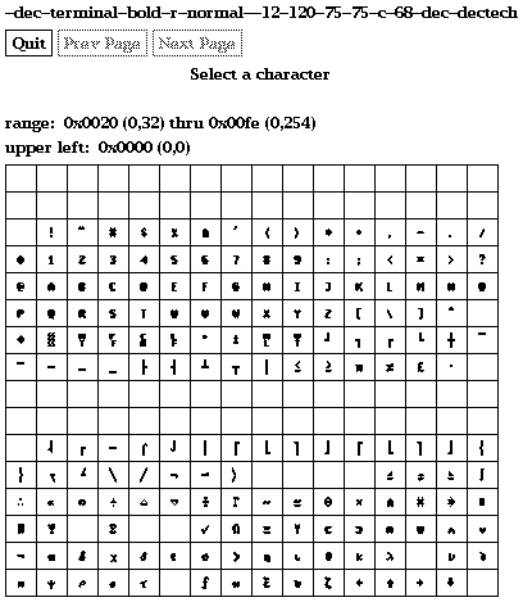


Figure 2-5. dec-dectech encoding

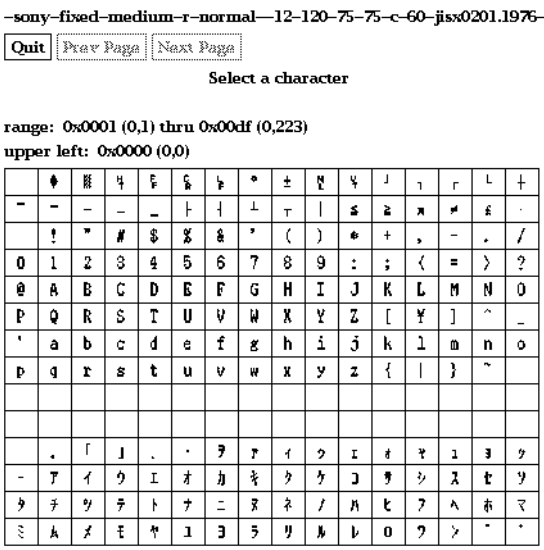


Figure 2-6. JIS0201-1976 Encoding

B

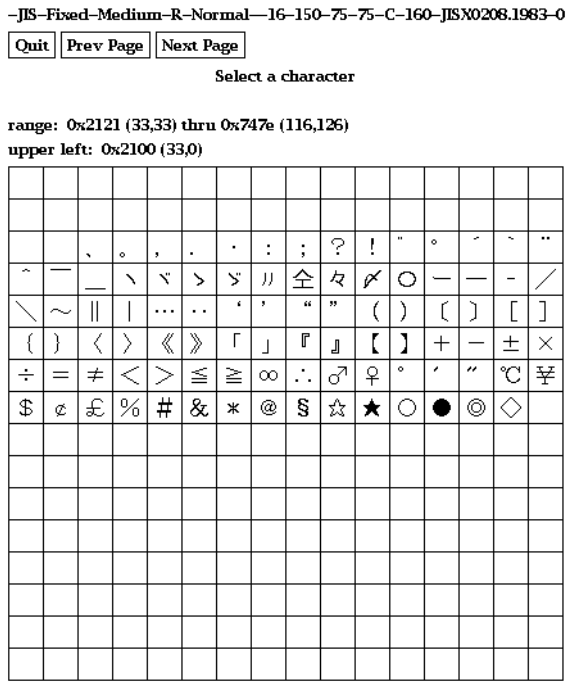


Figure 2-7. jisx0208.1983 Encoding

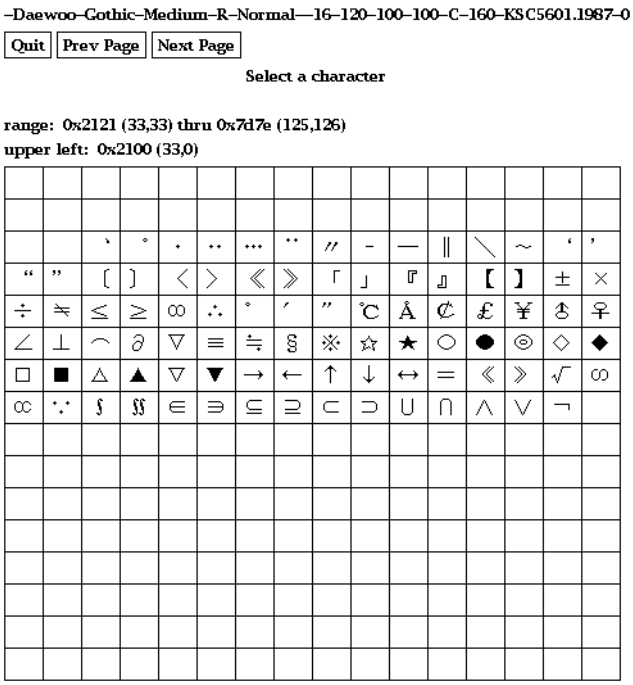


Figure 2-8. ksc5601.1987 Encoding

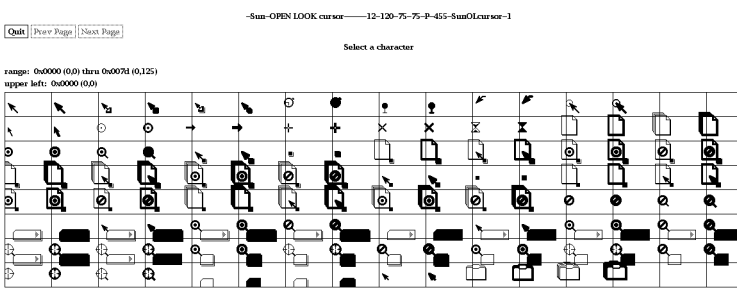


Figure 2-9. Sun OPEN LOOK Cursor Encoding

B

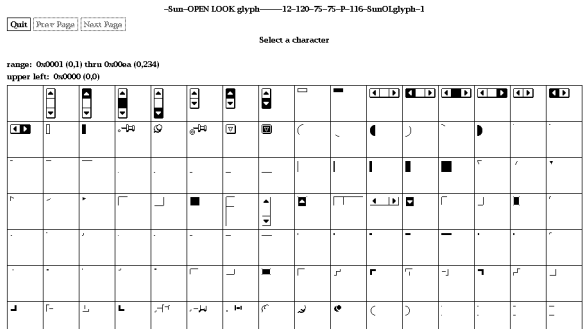


Figure 2-10. Sun OPEN LOOK Glyph Encoding

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

C

APPENDIX C

Standard Bitmaps for X11, OLIT and XView

This appendix shows the bitmaps included with the standard distribution of the X Window System. These can be used for setting window background, cursor symbols, pixmaps, and possibly for application icon pixmaps.

A number of bitmaps are included with the standard distribution of the X Window System. These bitmaps can be used for setting window background pixmaps and possibly for application icon pixmaps.

The standard bitmaps are generally located in the directory */usr/include/X11/bitmaps* . Each bitmap is in standard X11 bitmap format in its own file. The *bitmap* application can be used to view these bitmaps in larger scale and to edit them (though their permissions normally do not allow overwriting).

You can use these bitmaps to set the background pattern of a window in any application that allows it. For example, if you wanted to change the root window background pixmap, you could do so using *xsetroot* :

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

Note that the bitmaps that come in pairs, such as `cntr_ptr` and `cntr_ptrmsk`, are intended for creating pointer shapes. See Chapter 14, *Customization Clients*, for information on specifying a bitmap as the root window pointer.

The 63 bitmaps pictured on the following pages are included in the MIT Release 4 standard distribution of X. Table C-1 lists those bitmaps that have been added to the standard distribution in X11 Release 4.

Table C-1Standard Bitmaps Available in the MIT distribution

calculator	dropbar7	dropbar8

Table C-1Standard Bitmaps Available in the MIT distribution

escherknot	hlines2	hlines3
keyboard16	letters	mailempty
mailemptymsk	mailfull	mailfullmsk
menu10	menu12	menu16
menu8	noletters	plaid
terminal	vlines2	vlines3 xlogo11

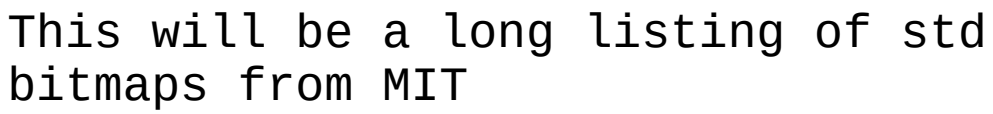


Figure C-1. The standard bitmaps

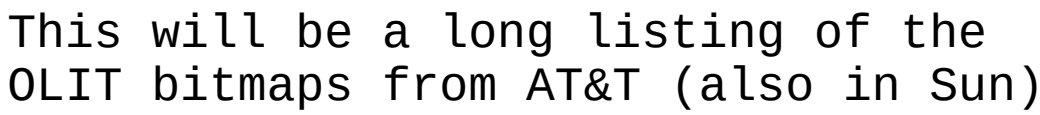


Figure C-2. The OLIT Bitmaps

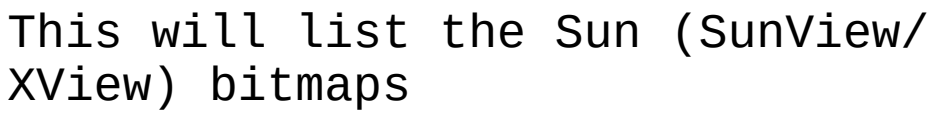


Figure C-3. The SunView/XView bitmaps

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX D

Standard Cursors

D.1 Cursors

This appendix shows the standard cursor images that can be used by X programs.

Tsble 1 lists the cursors available in the standard distribution of X from MIT; the cursor shapes themselves are pictured in Figure 1. The OPEN LOOK cursors are shown in Figure 2.

To specify a cursor as an argument to a command line option, as the value of a resource variable, etc., strip the `XC_` prefix from the symbol name. For example, to specify the `XC_sailboat` cursor as the *xterm* pointer, you could enter the command:

```
% xterm -xrm 'xterm*pointerShape: sailboat'
```

Each cursor has an associated numeric value (to the right of the symbol name in the table). You may notice that the values skip the odd numbers. Each cursor is actually composed of two font characters: the character that defines the shape (pictured in .Ref f), p and a mask character (not shown) that sets the cursor shape off from the root (or other) window. (More precisely, the mask selects which pixels in the screen around the cursor are disturbed by the cursor.) The mask is generally the same shape as the character it underlies but is one pixel wider in all directions.

To get an idea of what masks look like, display the entire cursor font using the command:

```
% xfd -fn cursor
```

The *olwm* window manager uses several of the standard OPEN LOOK cursor symbolsshow in Figure 2.

Table D-1. Standard Cursor Shapes

Symbol	Value	Symbol	Value
XC_X_cursor	0	XC_ll_angle	76
XC_arrow	2	XC_lr_angle	78
XC_based_arrow_down	4	XC_man	80
XC_based_arrow_up	6	XC_middlebutton	82
XC_boat	8	XC_mouse	84
XC_bogosity	10	XC_pencil	86
XC_bottom_left_corner	12	XC_pirate	88
XC_bottom_right_corner	14	XC_plus	90
XC_bottom_side	16	XC_question_arrow	92
XC_bottom_tee	18	XC_right_ptr	94
XC_box_spiral	20	XC_right_side	96
XC_center_ptr	22	XC_right_tee	98
XC_circle	24	XC_rightbutton	100
XC_clock	26	XC_rtl_logo	102
XC_coffee_mug	28	XC_sailboat	104
XC_cross	30	XC_sb_down_arrow	106
XC_cross_reverse	32	XC_sb_h_double_arrow	108
XC_crosshair	34	XC_sb_left_arrow	110
XC_diamond_cross	36	XC_sb_right_arrow	112
XC_dot	38	XC_sb_up_arrow	114
XC_dotbox	40	XC_sb_v_double_arrow	116
XC_double_arrow	42	XC_shuttle	118
XC_draft_large	44	XC_sizing	120
XC_draft_small	46	XC_spider	122
XC_draped_box	48	XC_spraycan	124
XC_exchange	50	XC_star	126
XC_fleur	52	XC_target	128

Table D-1. Standard Cursor Shapes

Symbol	Value	Symbol	Value
XC_gobbler	54	XC_tcross	130
XC_gumby	56	XC_top_left_arrow	132
XC_hand1	58	XC_top_left_corner	134
XC_hand2	60	XC_top_right_corner	136
XC_heart	62	XC_top_side	138
XC_icon	64	XC_top_tee	140
XC_iron_cross	66	XC_trek	142
XC_left_ptr	68	XC_ul_angle	144
XC_left_side	70	XC_umbrella	146
XC_left_tee	72	XC_ur_angle	148
XC_leftbutton	74	XC_watch	150
XC_xterm	152		

Insert display of std cursors from Generic Volume 3

Figure D-1. Standard Cursors

Insert display of OL cursors

Figure D-2. The OPEN LOOK Cursors

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX E

cmdtool and xterm Control Sequences

This appendix lists the escape sequences that can be used to control features of a *cmdtool* or *xterm* terminal emulator window.

A standard terminal performs many operations in response to escape sequences sent out by a program. In emulating a terminal, *cmdtool* or *xterm* responds to those same terminal escape sequences. Under UNIX, programs use the *termcap* or *terminfo* database to determine which escape sequences to send out. For more information, see the standard UNIX man pages *termcap* (5) or *terminfo* (5), or the Nutshell Handbook *Termcap and Terminfo*, available from O'Reilly & Associates, Inc.

E.1 cmdtool/shelltool Control Sequences

Since *cmdtool* was written as a window terminal, not as an emulator of a real terminal, it has little "baggage" to carry around; accordingly, the commands interpreted by *cmdtool*/*shelltool* are few in number, but more than adequate for most purposes. Table E-1 lists them. Part of this table is adapted from the *shelltool* manual page.

Table E-1. Cmdtool/shelltool escape sequences

Sequence	Function
BEL (Ctrl-G)	Make beeping sound
BS (Ctrl-H)	Erase char to left of cursor
TAB (Ctrl-I)	Horizontal tab
LF (Ctrl-J)	Newline; UNIX maps to "Return"
CR (Ctrl-M; Return)	Normal end-of-line character.

Table E-1. Cmdtool/shelltool escape sequences

Sequence	Function
ESC [1 t	open (de-iconify)
ESC [2 t	close (iconify)
ESC [3 t	move, with interactive feedback
ESC [3 ; TOP ; LEFT t	move window to TOP LEFT (pixel coordi- nates)
ESC [4 t	stretch, with interactive feed- back
ESC [4 ; HT ; WIDTH t	resize window, to HT WIDTH size (in pixels)
ESC [5 t	move window to front of window stack
ESC [6 t	move window to back of screen
ESC [7 t	refresh
ESC [8 ; ROWS ; COLS t	resize window, to ROWS COLS size (in characters)
ESC [11 t	report if window is open or iconic by sending ESC [1 t or ESC [2 t
ESC [13 t	report position by sending ESC [3 ; TOP ; LEFT t
ESC [14 t	report size in pixels by sending ESC [4 ; HT ; WIDTH t
ESC [18 t	report size in characters by sending ESC [8 ; ROWS ; COLS t
ESC [20 t	report icon label by sending ESC] L label ESC \
ESC [21 t	report tool header by sending ESC] l label ESC \
ESC] l text ESC \	set tool header to text
ESC] I file ESC \	set icon to the icon contained in file; file must be in iconedit output format
ESC] L label ESC \	set icon label to label

Table E-1. Cmdtool/shelltool escape sequences

Sequence	Function
ESC [> OPT ; ... h	turn SB OPT on (OPT = 1 => page-mode), for example, ESC [> 1 ; 3 ; 4h
ESC [> OPT ; ... k	report option OPT; sends ESC [> OPT l or ESC [> OPT h for each OPT
ESC [> OPT ;... l	turn option OPT off (OPT = 1 => pagemode), for ESC [> 1 ; 3 ;

E.2 xterm Control Sequences

Since *xterm* has to retain compatiibility with not one but two different terminal types, it has a vast array of escape sequences. This appendix purports to contain a complete list.

This appendix is based on two sources: the “Xterm Control Sequences” document, written by Edward Moy, University of California, Berkeley, for the X10 *xterm* ; and X11 updates provided to the X Consortium by Skip Montanaro, GE Corporate Research & Development.

E.2.1 Definitions

Most of these control sequences are standard VT102 control sequences. There are, however, additional ones to provide control of *xterm* -dependent functions, like the scrollbar or window size.

C A single (required) character.

Ps A single (usually optional) numeric parameter, composed of one of more digits.

Pm A multiple numeric parameter composed of any number of single numeric parameters, separated by ; character(s).

Pt A text parameter composed of printable characters.

E.2.2 VT102 Mode

Most of these control sequences are standard VT102 control sequences. There are, however, additional ones to provide control of xterm-dependent functions, like the scrollbar or window size.

BEL Bell (Ctrl-G)

BS Backspace (Ctrl-H)

TAB Horizontal Tab (Ctrl-I)

LF Line Feed or New Line (Ctrl-J)

VT Vertical Tab (Ctrl-K)

FF Form Feed or New Page (Ctrl-L)

CR Carriage Return (Ctrl-M)

SO Shift Out (Ctrl-N) -> Switch to Alternate Character Set

SI Shift In (Ctrl-O) -> Switch to Standard Character Set

ESC BEL Same as non-escaped BEL

ESC BS Same as non-escaped BS

ESC HT Same as non-escaped HT

ESC NL Same as non-escaped NL

ESC VT Same as non-escaped VT

ESC NP Same as non-escaped NP

ESC CR Same as non-escaped CR

ESC SO Same as non-escaped SO

ESC SI Same as non-escaped SI

ESC #BEL Same as non-escaped BEL

ESC #BS Same as non-escaped BS

ESC #HT Same as non-escaped HT

ESC #NL Same as non-escaped NL

ESC #VT Same as non-escaped VT

ESC #NP Same as non-escaped NP

ESC #CR Same as non-escaped CR

ESC #SO Same as non-escaped SO

ESC #SI Same as non-escaped SI

ESC #8 DEC Screen Alignment Test (DECALN)

ESC (BEL Same as non-escaped BEL

ESC (BS Same as non-escaped BS

ESC (HT Same as non-escaped HT

ESC (NL Same as non-escaped NL

ESC (VT Same as non-escaped VT

ESC (NP Same as non-escaped NP

ESC (CR Same as non-escaped CR

ESC (SO Same as non-escaped SO

ESC (SI Same as non-escaped SI

ESC (C Select G0 Character Set (SCS)

C = 0 -> Special Character and Line Drawing Set

C = 1 -> Alternate Character ROM Standard Set

C = 2 -> Alternate Character ROM Special Set

C = A -> United Kingdom (UK)

C = B -> United States (USASCII)

ESC)C Select G1 Character Set (SCS)

C = 0 -> Special Character and Line Drawing Set

C = 1 -> Alternate Character ROM Standard Set

C = 2 -> Alternate Character ROM Special Set

C = A -> United Kingdom (UK)

C = B -> United States (USASCII)

ESC *C Select G2 Character Set (SCS)

C = 0 -> Special Character and Line Drawing Set

C = 1 -> Alternate Character ROM Standard Set

C = 2 -> Alternate Character ROM Special Set

C = A -> United Kingdom (UK)

C = B -> United States (USASCII)

ESC +C Select G3 Character Set (SCS)

C = 0 -> Special Character and Line Drawing Set

C = 1 -> Alternate Character ROM Standard Set

C = 2 -> Alternate Character ROM Special Set

C = A -> United Kingdom (UK)

C = B -> United States (USASCII)

ESC 7 Save Cursor (DECSC)

ESC 8 Restore Cursor (DECRC)

ESC = Application Keypad (DECPAM)

ESC > Normal Keypad (DECPNM)

ESC D Index (IND)

ESC E Next Line (NEL)

ESC H Tab Set (HTS)

ESC M Reverse Index (RI)

ESC N Single Shift Select of G2 Character Set (SS2)

ESC O Single Shift Select of G3 Character Set (SS3)

ESC Return Terminal ID (DECID)

ESC [BEL Same as non-escaped BEL

ESC [BS Same as non-escaped BS

ESC [HT Same as non-escaped HT

ESC [NL Same as non-escaped NL

ESC [VT Same as non-escaped VT

ESC [NP Same as non-escaped NP

ESC [CR Same as non-escaped CR

ESC [SO Same as non-escaped SO

ESC[SI Same as non-escaped SI

ESC [?BEL Same as non-escaped BEL

ESC [?BS Same as non-escaped BS

ESC [?HT Same as non-escaped HT

ESC [?NL Same as non-escaped NL

ESC [?VT Same as non-escaped VT

ESC [?NP Same as non-escaped NP

ESC [?CR Same as non-escaped CR

ESC [?SO Same as non-escaped SO

ESC [?SI Same as non-escaped SI

ESC [Ps@ Insert Ps (Blank) Character(s) (default = 1) (ICH)

ESC [PsA Cursor Up Ps Times (default = 1) (CUU)

ESC [PsB Cursor Down Ps Times (default = 1) (CUD)

ESC [PsC Cursor Forward Ps Times (default = 1) (CUF)

ESC [PsD Cursor Backward Ps Times (default = 1) (CUB)

ESC [Ps;PsH Cursor Position [row;column] (default = [1,1]) (CUP)

ESC [PsJ Erase in Display (ED)

Ps = 0 -> Clear Below (default)

Ps = 1 -> Clear Above

Ps = 2 -> Clear All

ESC [PsK Erase in Line (EL)

Ps = 0 -> Clear to Right (default)

Ps = 1 -> Clear to Left

Ps = 2 -> Clear All

ESC [PsL Insert Ps Line(s) (default = 1) (IL)

ESC [PsM Delete Ps Line(s) (default = 1) (DL)

ESC [PsP Delete Ps Character(s) (default = 1) (DCH)

ESC [T Track mouse

ESC [Psc Device Attributes (DA1)

ESC [Ps;Psf Cursor Position [row;column] (default = [1,1]) (HVP)

ESC [Psg Tab Clear

Ps = 0 -> Clear Current Column (default)

Ps = 3 -> Clear All

ESC[Psh Mode Set (SET)

Ps = 4 -> Insert Mode (IRM)

Ps = 20 -> Automatic Linefeed (LNM)

ESC [Psl Mode Reset (RST)

Ps = 4 -> Insert Mode (IRM)

Ps = 20 -> Automatic Linefeed (LNM)

ESC [Pmm Character Attributes (SGR)

Pm = 0 -> Normal (default)

Pm = 1 -> Blink (appears as Bold)

Pm = 4 -> Underscore

Pm = 5 -> Bold

Pm = 7 -> Inverse

ESC [Psn Device Status Report (DSR)

Ps = 5 -> Status Report ESC[0n -> OK

Ps = 6 -> Report Cursor Position (CPR) [row;column] as ESC[r;cR

ESC [Ps;Psr Set Scrolling Region [top;bottom] (default = full size of window) (DECSTBM)

ESC [Psx Request Terminal Parameters (DECREQTPARM)

ESC [Ps ND string NP

OSC Mode

ND can be any non-digit Character (it's discarded)

NP can be any non-printing Character (it's discarded)

string can be any ASCII printable string (max 511 characters)

Ps = 0 -> use string as a new icon name and title

Ps = 1 -> use string as a new icon name only

Ps = 2 -> use string as a new title only

Ps = 46 -> use string as a new log file name

ESC [?Psh DEC Private Mode Set (DECSET)

Ps = 1 -> Application Cursor Keys (DECCKM)

Ps = 2 -> Set VT52 Mode

Ps = 3 -> 132 Column Mode (DECCOLM)

Ps = 4 -> Smooth (Slow) Scroll (DECSCLM)

Ps = 5 -> Reverse Video (DECSCNM)

Ps = 6 -> Origin Mode (DECOM)

Ps = 7 -> Wraparound Mode (DECAWM)

Ps = 8 -> Auto-repeat Keys (DECARM)

Ps = 9 -> Send MIT Mouse Row & Column on Button Press

Ps = 38 -> Enter TekTronix Mode (DECTEK)

Ps = 40 -> Allow 80 <--> 132 Mode

Ps = 41 -> curses(5) fix

Ps = 44 -> Turn On Margin Bell

Ps = 45 -> Reverse-wraparound Mode

Ps = 46 -> Start Logging

Ps = 47 -> Use Alternate Screen Buffer

Ps = 1000 -> send VT200 Mouse Row & Column on Button Press

Ps = 1003 -> send VT200 Hilite Mouse Row & Column on Button Press

ESC [?Psl DEC Private Mode Reset (DECRST)

Ps = 1 -> Normal Cursor Keys (DECCKM)

Ps = 3 -> 80 Column Mode (DECCOLM)

Ps = 4 -> Jump (Fast) Scroll (DECSCLM)

Ps = 5 -> Normal Video (DECSCNM)

Ps = 6 -> Normal Cursor Mode (DECOM)

Ps = 7 -> No Wraparound Mode (DECAWM)

Ps = 8 -> No Auto-repeat Keys (DECARM)

Ps = 9 -> Don’t Send MIT Mouse Row & Column on Button Press

Ps = 40 -> Disallow 80 <--> 132 Mode

Ps = 41 -> No curses(5) fix

Ps = 44 -> Turn Off Margin Bell

Ps = 45 -> No Reverse-wraparound Mode

Ps = 46 -> Stop Logging

Ps = 47 -> Use Normal Screen Buffer

Ps = 1000 -> Don’t send Mouse Row & Column on Button Press

Ps = 1003 -> Don’t send Hilite Mouse Row & Column on Button Press

ESC [?Psr Restore DEC Private Mode

Ps = 1 -> Normal/Application Cursor Keys (DECCKM)

Ps = 3 -> 80/132 Column Mode (DECCOLM)

Ps = 4 -> Jump (Fast)/Smooth (Slow) Scroll (DECSCLM)

Ps = 5 -> Normal/Reverse Video (DECSCNM)

Ps = 6 -> Normal/Origin Cursor Mode (DECOM)

Ps = 7 -> No Wraparound/Wraparound Mode (DECAWM)

Ps = 8 -> Auto-repeat/No Auto-repeat Keys (DECARM)

Ps = 9 -> Don’t Send/Send MIT Mouse Row & Column on Button Press

Ps = 40 -> Disallow/Allow 80 <--> 132 Mode

Ps = 41 -> Off/On curses(5) fix

Ps = 44 -> Turn Off/On Margin Bell

Ps = 45 -> No Reverse-wraparound/Reverse- wraparound Mode

Ps = 46 -> Stop/Start Logging

Ps = 47 -> Use Normal/Alternate Screen Buffer

Ps = 1000 -> Don't send/send VT200 Mouse Row &Column onButton Press

Ps = 1003 -> Don't send/send VT200 HiliteMouse Row & Column on Button Press

ESC [?Pss Save DEC Private Mode

Ps = 1 -> Normal/Application Cursor Keys (DECCKM)

Ps = 3 -> 80/132 Column Mode (DECCOLM)

Ps = 4 -> Jump (Fast)/Smooth (Slow) Scroll (DECSCLM)

Ps = 5 -> Normal/Reverse Video (DECSCNM)

Ps = 6 -> Normal/Origin Cursor Mode (DECOM)

Ps = 7 -> No Wraparound/Wraparound Mode (DECAWM)

Ps = 8 -> Auto-repeat/No Auto-repeat Keys (DECARM)

Ps = 9 -> Don't Send/Send MIT Mouse Row & Column on Button Press

Ps = 40 -> Disallow/Allow 80 <--> 132 Mode

Ps = 41 -> Off/On curses(5) fix

Ps = 44 -> Turn Off/On Margin Bell

Ps = 45 -> No Reverse-wraparound/Reverse-wraparound Mode

Ps = 46 -> Stop/Start Logging

Ps = 47 -> Use Normal/Alternate Screen Buffer

Ps = 1000 -> Don't send/send VT200 Mouse Row & Column on Button Press

Ps = 1003 -> Don't send/send VT200 Hilite Mouse Row & Column on Button Press

ESC]Ps;PtBEL

Set Text Parameters

Ps = 0 -> Change Window Name and Title to Pt

Ps = 1 -> Change Window Name to Pt

Ps = 2 -> Change Window Title to Pt

Ps = 46 -> Change Log File to Pt

Ps = 50 -> Change Font to Pt

ESC c Full Reset (RIS)

ESC n Locking Shift Select of G2 Character Set (LS2)

ESC Locking Shift Select of G3 Character Set (LS3)

E.2.3 Tektronix 4014 Mode

Most of these sequences are standard Tektronix 4014 control sequences. The major features missing are the alternate (APL) character set and the write-thru and defocused modes.

BEL Bell (Ctrl-G)

BS Backspace (Ctrl-H)

TAB Horizontal Tab (Ctrl-I)

LF Line Feed or New Line (Ctrl-J)

VT Vertical Tab (Ctrl-K)

FF Form Feed or New Page (Ctrl-L)

CR Carriage Return (Ctrl-M)

ESC ETX Switch to VT102 Mode

ESC ENQ Return Terminal Status

ESC LF PAGE (Clear Screen)

ESC ETB COPY (Save Tektronix Codes to File)

ESC CAN Bypass Condition

ESC SUB GIN mode

ESC FS Special Point Plot Mode

ESC GS Graph Mode (same as GS)

ESC RS Incremental Plot Mode (same as RS)

ESC US Alpha Mode (same as US)

ESC 8 Select Large Character Set

ESC 9 Select #2 Character Set

ESC : Select #3 Character Set

ESC ; Select Small Character Set

ESC]Ps;PtBEL

Set Text Parameters

Ps = 0 -> Change Window Name and Title to Pt

Ps = 1 -> Change Icon Name to Pt

Ps = 2 -> Change Window Title to Pt

Ps = 46 -> Change Log File to Pt

ESC ` Normal Z Axis and Normal (solid) Vectors

ESC a Normal Z Axis and Dotted Line Vectors

ESC b Normal Z Axis and Dot-Dashed Vectors

ESC c Normal Z Axis and Short-Dashed Vectors

ESC d Normal Z Axis and Long-Dashed Vectors

ESC h Defocused Z Axis and Normal (solid) Vectors

ESC i Defocused Z Axis and Dotted Line Vectors

ESC j Defocused Z Axis and Dot-Dashed Vectors

ESC k Defocused Z Axis and Short-Dashed Vectors

ESCl Defocused Z Axis and Long-Dashed Vectors

ESC p Write-Thru Mode and Normal (solid) Vectors

ESC q Write-Thru Mode and Dotted Line Vectors

ESC r Write-Thru Mode and Dot-Dashed Vectors

ESC s Write-Thru Mode and Short-Dashed Vectors

ESC t Write-Thru Mode and Long-Dashed Vectors

FS Point Plot Mode

GS Graph Mode

RS Incremental Plot Mode

US Alpha Mode

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX F

Translation Table Syntax

This appendix describes the basic syntax of translation table resources, described in Chapter 12, *Setting Resources*.

This appendix explains some of the more complex aspects of translation table syntax. It probably gives more detail than the average user will need but we've included it to help clarify this rather complicated topic.

F.1 Event Types and Modifiers

The syntax of the translation table is sufficiently general to encompass a wide variety of events and circumstances. Event translations can be specified to handle characteristic user interface idioms like double clicking, dragging, or combining keyboard modifiers with pointer button input. To specify translations that use these features, it is necessary to learn more about the detailed syntax used to specify translations.

An activity susceptible to translation is a sequence of events and modifiers (that perform an action). Events are specified in angle brackets and modifiers precede the event they modify. The legal events that can be specified in a translation are as shown in Table F-1.

Table F-1. Event Types and Their Abbreviations

Event Name	Event Type	Abbreviations/Synonyms
KeyPress	Keyboard	Key , KeyDown
KeyUp	Keyboard	KeyRelease
ButtonPress	Mouse Button	BtnDown
ButtonRelease	Mouse Button	BtnUp

Table F-1. Event Types and Their Abbreviations

Event Name	Event Type	Abbreviations/Synonyms
Btn1Down	Mouse Button Press	
.		
.		
Btn5Down		
Btn1Up	Mouse Button Release	
.		
.		
Btn5Up		
MotionNotify	Mouse Motion	Motion , MouseMoved,PtrMoved
ButtonMotion	Motion w/any Button Down	BtnMotion
Button1Motion	Motion w/Button Down	Btn1Motion
.		.
.		.
Button5Motion		Btn5Motion
EnterNotify	Mouse in Window	Enter, EnterWindow
LeaveNotify		LeaveWindow , Leave
FocusIn	Keyboard Input Focus	
FocusOut		
KeymapNotify	Changed Key Map	Keymap
ColormapNotify	Changed Color Map	Clrmap
Expose	Related Exposure Events	
GraphicsExpose		GrExp
NoExpose		NoExp
VisibilityNotify		Visible
CreateNotify	Window Management	Create

Table F-1. Event Types and Their Abbreviations

Event Name	Event Type	Abbreviations/Synonyms
DestroyNotify		Destroy
UnmapNotify		Unmap
MapNotify		Map
MapRequest		MapReq
ReparentNotify		Reparent
ConfigureNotify		Configure
ConfigureRequest		ConfigureReq
GravityNotify		Grav
ResizeRequest		ResReq
CirculateNotify		Circ
CirculateRequest		CircReq
PropertyNotify		Prop
SelectionClear	Intra-client Selection	SelClr
SelectionRequest		SelReq
SelectionNotify		Select

The possible modifiers of an event are listed in the table. The modifiers Mod1 through Mod5 are highly system-dependent, and may not be implemented by all servers.

Table F-2. Key Modifiers

Event Modifiers	Abbreviation
Ctrl	c
Meta	m
Shift	s
Lock	l
Any	
ANY	

Table F-2. Key Modifiers

Event Modifiers	Abbreviation
None	
Mod1	1
.	.
Mod5	5

F.1.1 Detail Field

To provide finer control over the translation process, the event part of the translation can include an additional "detail." For example, if you want the event to require an additional keystroke, for instance, an A key, or a Control-T, then that keystroke can be specified as a translation detail. The default detail field is *ANY* .

The valid translation details are event-dependent. For example, to specify the above example for keypress events, you would use:

```
<Key>A
```

and:

```
Ctrl<Key>T
```

respectively.

Key fields can be specified by the keysym value, as well as by the keysym symbolic name. For example, the keysym value of the Delete key is 0xffff. Keysym values can be determined by examining the file *<X11/keysymdef.h>* or by using the *xmodmap* client. (See Chapter 14, *Customization Clients*). Unfortunately, with some translations the keysym value may actually be required, since not all keysym symbolic names may be properly interpreted.

F.1.2 Modifiers

Modifiers can be closely controlled to define exactly which events can be specified. For example, if you want the action to be performed by pointer button clicks but not by pointer button clicks with the Control or Shift key down, these limitations can be specified. Similarly, if you don't care if there are modifiers present, this can also be specified.

Table F-3 lists the available event modifiers.

Table F-3. Event Modifiers and their Meanings

Modifier	Meaning
None *<event>*	No modifiers allowed.
<event>	Doesn't care. Any modifiers okay.

Table F-3. Event Modifiers and their Meanings

Modifier	Meaning
Mod1 Mod	Mod1 and Mod2, plus any others (i.e., anything that includes m1 and m2).
!Mod1 Mod2<*event*>	Mod1 and Mod2 but nothing else.
Mod1 ~Mod2<*event*>	Mod1 and not Mod2.

F.1.3 Complex Translation Examples

The following translation specifies that function *f* is to be invoked when both the Shift key and the third pointer button are pressed.

```
Shift<Btn3Down>: f()
```

To specify that both the Control and Shift keys are to be pressed use:

```
Ctrl Shift<Btn3Down>: f()
```

To specify an optional repeat count for an activity, put a number in parentheses after the action. The number refers to the whole translation. To make the last example require a double-click, with both Control and Shift keys pressed, use:

```
Ctrl Shift<Btn3Down>(2): f()
```

The server distinguishes between single-clicks and double-clicks based on a pre-programmed timing interval. If a second click occurs before the interval expires, then the event is interpreted as a double-click; otherwise the event is interpreted as two single-clicks. The variable `clickTime` is maintained deep in the internals of X. Unfortunately, thus far there is no way to set this time interval to match user preference. Currently it is set to be 200 milliseconds.

A translation involving two or more clicks can be specified as (2+) in the previous example. In general, a plus sign following the number *n* would mean *n* or more occurrences of the event.

Multiple events can be specified by separating them with commas on the translation line. To indicate pressing button 1, pressing button 2, then releasing button 1, and finally releasing button 2, use:

```
<Btn1Down>,<Btn2Down>,<Btn1Up>,<Btn2Up>: f()
```

Another way to describe this action in English would be to say “while button 1 is down, click button 2.” “Meaningless” pointer movement is generally ignored. In the previous case, for example, if pointer motion occurred while the buttons were down, it would not interfere with detection of the event. Thus, inadvertent pointer jiggling will not thwart even the most complex user-input sequences.

F

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX G

Introduction to Xt Widget Resources

We've said several times that you can set "resources" to control the behavior of client programs. Unfortunately, to make full use of this feature, you need to know a little about the program you want to customize. If the program is an XView application—the OpenWindows DeskSet, *cmdtool*, and most current Sun programs—it uses a simpler hierarchy of resources. If the program is written using any "Xt Intrinsices" or "Widget"-based toolkit—such as OLIT, Motif, or the MIT Athena Widgets—then you need to know a little more about the individual widgets. These toolkits do allow you to set not only resources defined by the application itself, but also resources that apply to any of the widgets that make up the application. The reference page for the application sometimes lists the most important of these resources, but for fuller customization, you need to know more about each widget.

Unfortunately, the design of the X Toolkit is such that to really do the right thing, you probably need to know a bit more about Toolkit programming than the average user might like.

In this appendix we provide some introductory concepts about how widgets are used in X Toolkit programs. The following two appendices provide reference information about each class of widgets in the OpenWindows version of OLIT as well as the MIT Athena Widgets used in many MIT and contributed applications. If you are a Toolkit programmer or other sophisticated user, feel free to skip right to the widget reference descriptions.

G.1 The Widget Class Hierarchy

The first thing you need to know is how widgets are built.

Rather than starting each widget from scratch, the widget programmer starts with a copy of another, more basic widget, and modifies it. This process is called *subclassing* the widget, and the sequence of widgets leading up to the one you see is called its *class hierarchy*. Because of the way subclassing works, a widget *inherits* all of the characteristics of its superclasses, except those that are explicitly overridden or changed.

The class hierarchy starts with a class called Object, which defines some basic characteristics common to all widgets—namely the ability to understand resources, and to be linked to applications via a mechanism referred to as a callback. (When you click on a "quit" button, and the application quits, that is because the widget has communicated with the application via a callback.)

RectObj is a subclass of Object. RectObj adds various resources having to do with the fact that widgets are rectangular: width, height, borderWidth, and x, y positions. RectObj also adds resources for sensitivity—the fact that a widget can be temporarily "disabled" by a client (so for example, not allowing you to choose an option on a menu that would close a file if no file was open.)

Core is the first true widget in the class hierarchy. Object and RectObj don't have windows associated with them, and can never be "instantiated"—created and mapped to the screen. In fact, prior to Release 4, they were "invisible" even to Toolkit programmers, who simply assumed that Core was the root of the widget hierarchy.

The reason we now talk openly about Object and RectObj is that the R4 Toolkit supports a different class of object, known colloquially as a gadget, which is subclassed directly from RectObj, and does not have a window associated with it. It can be used only within a widget that understands how to manage gadgets, and allocates some of its own window space to display them. This is typically done when there are many identical widgets. (The only gadgets in the Athena widget set are the SmeBSB and SmeLine gadgets used to implement panes in a SimpleMenu widget. OLIT and Motif offer both widget and gadget versions of many of their objects, including all kinds of command buttons.)

At any rate, for most purposes, you can still act as though Core is the root of the widget hierarchy, since all widgets are subclassed from it, and therefore share all of its resources. The phrase "Core resources" is a fluke of terminology that can be misleading to new users. Because it sounds meaningful just as a general term, it isn't clear that the Core resources are really the resources of a particular widget class (rather than something magically recognized as central or "core" by the X Toolkit.)

Let's take a brief look at the some of the Core resources, which appear in Table G-1. The list includes the resources inherited from Object and RectObj, plus those added by Core.

Table G-1. Core Resources

background	Background	
borderColor	BorderColor	
borderWidth	BorderWidth	1
height	Height	0
width	Width	0
x	Position	0
y	Position	0

Some of these Core resources set obvious characteristics of a widget: `background` (color), `borderColor`, and `borderWidth` (in pixels). `height` and `width` specify the dimensions of the widget in pixels. `x` and `y` represent the x,y coordinates of the widget in relation to its parent.

Technically, `foreground` is *not* a Core resource. However, since `foreground` is defined by every widget class that does any drawing of lines or text, it can be assumed to be.

Note that Table G-1 isn't actually a complete list of all of the Core resources, but only of those that might be set by users. Some resources (such as callbacks) can only be set by programmers. The Toolkit doesn't even support a mechanism for understanding how to set them in a resource file. In addition, there are two resources that are so common that you might expect them to be Core resources, but which are not. They are defined individually by each of the widget classes that use them. This can be confusing, especially since they do correspond to standard X Toolkit options. But really, it is hair-splitting to worry about where they are defined—they are sufficiently standard to fall under the colloquial understanding of Core resources.

We'll say more about the subject of resource conversion later. First, though, let's finish describing the base classes provided by the X Toolkit, which are common to all Xt-based widget sets (including OLIT, Motif ,and the Athena widgets).

There is a special class of widgets whose job is to manage the size and/or position of other widgets. These are called Composite widgets, and all such geometry-managing widgets, are children of the Composite widget class. Composite inherits all of the characteristics of Core, and adds resources (settable only by the programmer) for identifying which widgets it should treat as its children.

Some simple geometry-managing widgets such as the Athena Box widget are direct subclasses of Core. However, there is another, more complex class of geometry-managing widget defined by the X Toolkit Intrinsics, called Constraint. A constraint widget defines special resources, called constraint resources, that apply to its children rather than to itself. They are actually resources of the constraint widget, but are specified as if they were resources of the child. The clearest example of constraint resources is provided by the Athena Form widget, which allows widgets to be positioned with respect to one another, so that they always keep the same arrangement, even when the Form is resized. For example, *xcalc* is implemented using a Form widget. Resources such as:

```
Form widget
    |   Command widget child of Form
    |   |   |   Constraint resource of Form seems to be resource of child
    |   |   |
XCalc.ti.button12.fromHoriz: button11
XCalc.ti.button12.fromVert: button7
```

specify that button12 (label PI) should always be next to button11 (label x!), and over button7 (label 7). We'll return to this example later, when we talk about the instance hierarchy of widgets in an application.

At any rate, there is one other subclass of Composite that bears mention: the Shell widget class. Shell widgets are simple composite widgets; they manage only one child—the application's main window, and they make themselves exactly the same size, so that they are hidden behind it. Even though you never see them, though, Shell widgets are extremely important, since they are the widgets that know how to interact with the window manager. Shell introduces several resources of importance to the application programmer, but only one of importance to the user: `geometry` (class `Geometry`).

There are actually seven subclasses of Shell, three of which are for internal toolkit purposes and four of which are used by application programmers in different circumstances. For example, there is one kind of shell widget used for the main window of an application (class TopLevelShell) and another used as the parent of a popup widget like a dialog box (class OverrideShell) that should never be manipulated by the window manager. (Notice that *mwm* doesn't reparent dialogs—they don't get a titlebar of their own, and can't be moved independently—this is because they are children of an OverrideShell, which overrides window manager intervention.)

There is another class of shell widget, called a TransientShell, which is used for popups, but can be manipulated by the window manager. An application might use a TransientShell for a popup help window that would be allowed to remain on the screen, and could be moved or resized separately (but not iconified.) An ApplicationShell is used by an application that has more than one completely independent window, as the class for its secondary "top level" windows.

For all practical purposes, you don't need this much information about shell widgets. As we'll see shortly, the only reference to a shell widget in a resource specification is typically via the application name, which the shell widget takes as its own.

Returning to widgets that you actually do see and interact with, let's consider the class derivation of a widget like Command, which is used to implement buttons you can click on with the mouse to ask the application to do something.

The Athena Command widget is a subclass of the Label widget, which is a subclass of the Simple widget, which in turn is a subclass of Core. As a result, Command inherits all of the Core resources, plus the resources of the Athena Simple widget (for practical purposes, the cursor that is to be displayed when the pointer is in the window), plus the resources of the Label widget—such as the ability to display a label, in a particular font. Command adds the ability (defined by the programmer, not the user) to call a particular application function when the button is clicked on.

Figure G-1 shows the complete class hierarchy of the Athena widgets used in all of the standard MIT applications described in this book. The widgets shown in gray are defined by the X Toolkit intrinsics, and are common to all Xt-based widget sets, including OLIT and Motif.

The *listres* application, without any arguments, lists the inheritance hierarchy for each of the Athena widgets. Given the name of any widget class, it lists all of the resources for that widget, and which superclass they are inherited from. Figure XXX shows listres for the Athena Label widget.: As we'll describe later in this appendix, not all of the resources

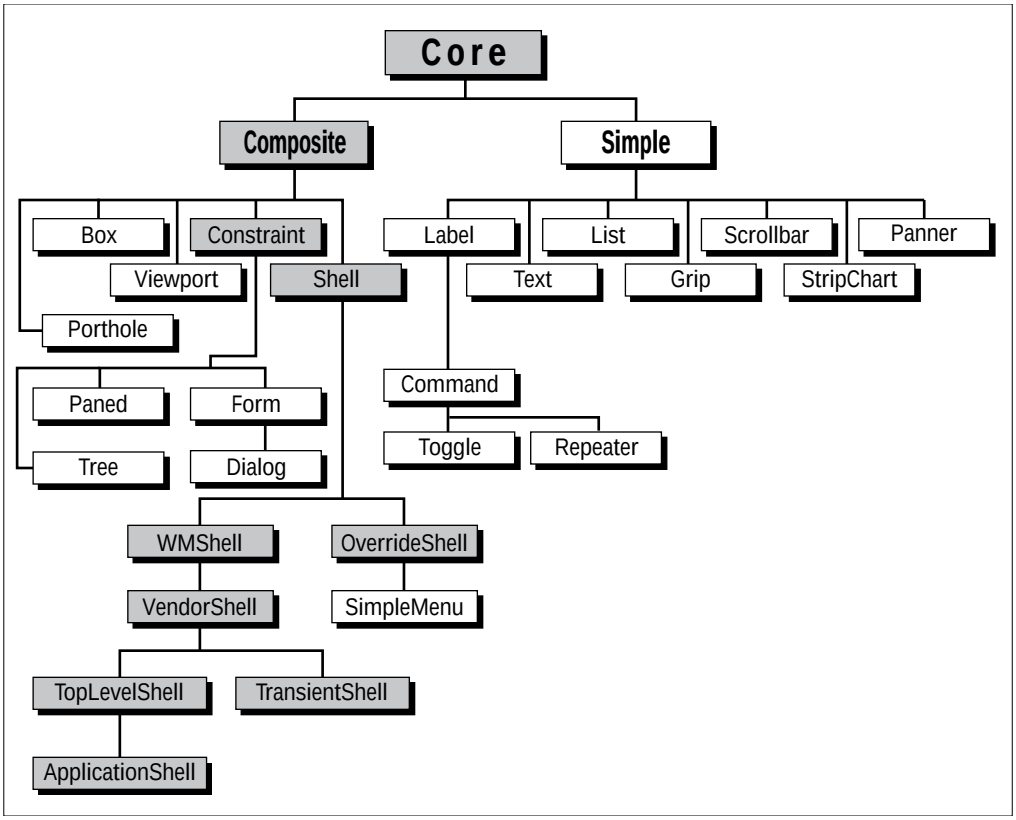


Figure G-1. Inheritance among the Athena widgets

listed by *listres* can be set in a resource file. However, this listing can provide a handy quick reference.

G.2 Widgets in the Application

Widget inheritance of resources from superclasses is an important part of the background to understanding how to affect the widget resources in the application, but it is not the whole story. Let's talk for a moment about how these widgets are used.

To make things more concrete, let's look at an actual application. *xclipboard* is a good choice. It uses several different widget classes, but isn't too complex. .Figure G-3 illustrates the widgets that make up *xclipboard*.

Every Toolkit application begins with a call to a function called `XtInitialize`, which in looks something like this[1]

```
top = XtInitialize( "xclipboard", "XClipboard", ... );
```

G

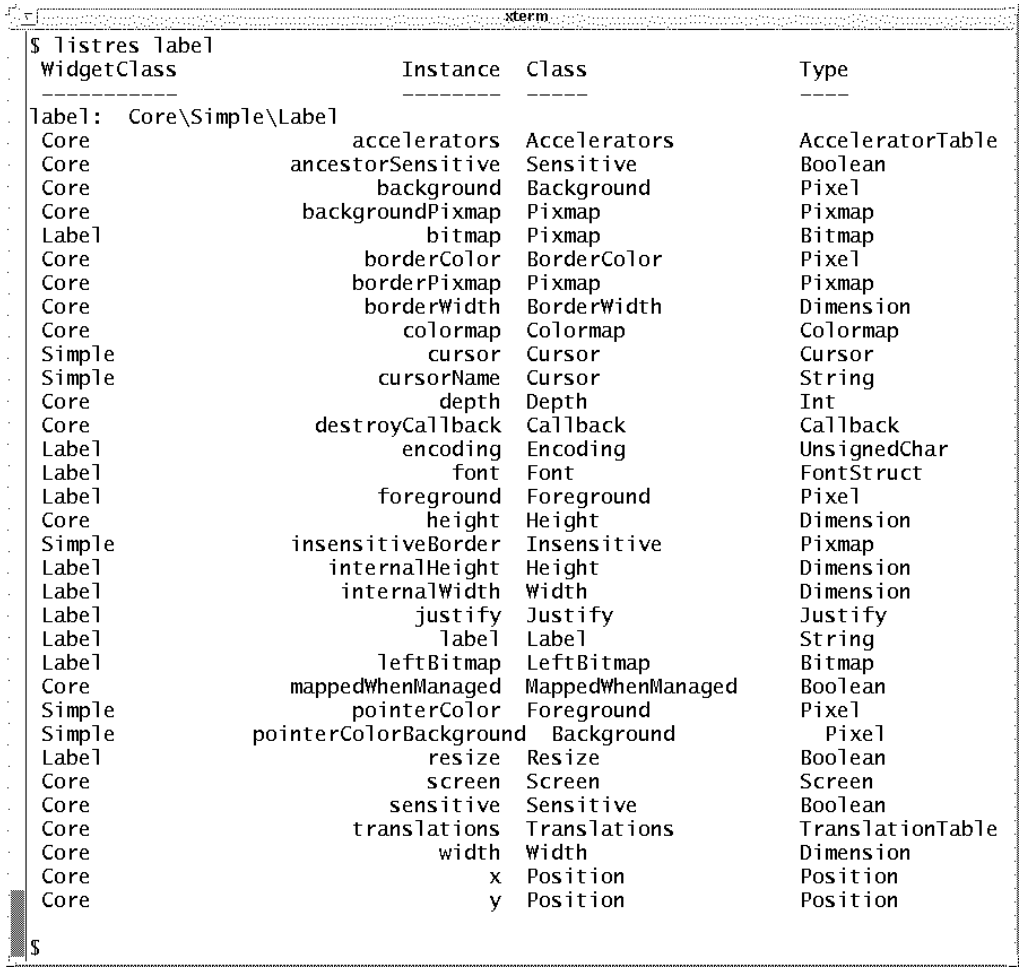


Figure G-2. listres for "label" widget

The first two arguments to this function give the instance name and class name of the application. This becomes the start of any resource specification for this application. And we know that if *xclipboard* has an app-defaults file, it will be called *XClipboard*, since that name is taken from the class name of the application. Notice that there's no magic here: this is under the explicit control of the application programmer. If he doesn't follow the conventions for the application's class and instance name, he'd better document what names he used here!

1. Actually, modern Toolkit applications are supposed to use the more complex `XtAppInitialize`, but `XtInitialize` makes the concept clearer, and besides, it's what *xclipboard* actually uses.

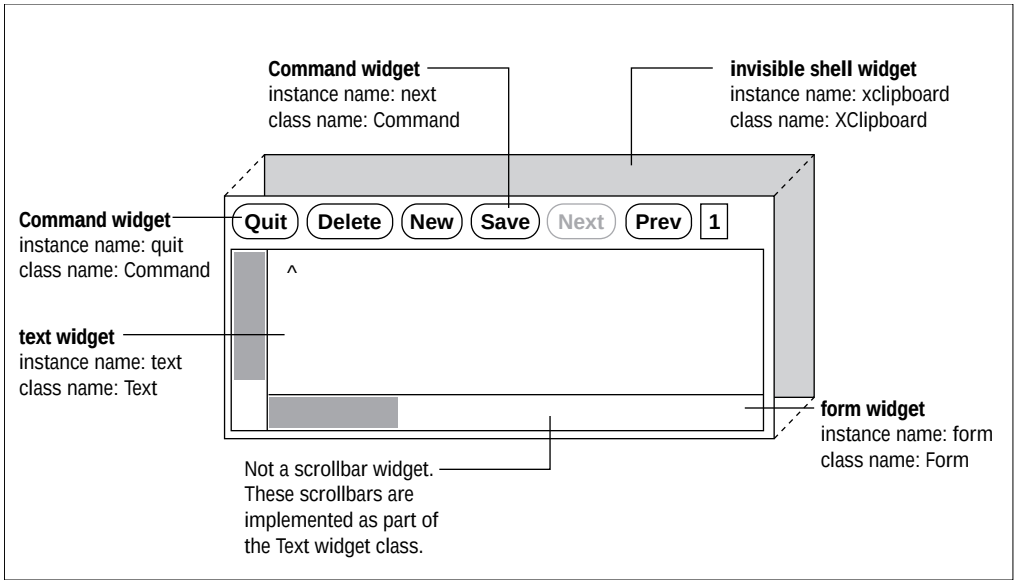


Figure G-3. Anatomy of an X Toolkit application

`XtInitialize`, among its other activities, creates a TopLevelShell widget. The name before the equals sign, `top`, is the name that the programmer will use to refer to this widget whenever he needs to use it in the application. This is completely irrelevant to the name the widget publishes for itself as its instance name.

Next, the program begins to create the widgets in the application, using a function called `XtCreateManagedWidget`. The first widget to be created is the main application widget, which in this case is a Form widget.

```
parent = XtCreateManagedWidget("form", formWidgetClass, top, ... );
```

The first argument to `XtCreateManagedWidget` is the instance name of the widget (`form`)—this is the name that will be used to refer to it in resource files. The second argument is a symbol identifying which widget class this widget should be.

Notice that the instance name is entirely arbitrary, and depends completely on the whim of the application programmer. Many applications that use only one instance of a widget class will give it an instance name that mirrors the class name, except in lower case, as was done here. But you can see that the programmer could just as well have given the widget the instance name "foo" or "main" or "grandma_moses." The implication is that unless the client's man page documents a widget instance name, you won't know what to use in a resource file.[1]

1. As of R4, all of the MIT client reference pages do list the instance names of all the widgets in the application.

The class name, on the other hand, is a part of the definition of a widget's class. It is always the same.

The third argument is the widget's parent—the geometry-managing widget that this widget will be displayed inside, and which will control its size and position. Notice that the parent of the form is `top`—the shell widget created by `XtInitialize`. As noted earlier, Shell widgets take just one child, and resize themselves so they fit completely behind that child, and are invisible.

Remember, though, that the program's internal name for the shell widget is not important when it comes to resource specifications. The Shell widget takes its "resource name and class" from the `XtInitialize` call.

If you're following the flow of the argument, you can see that to refer to this widget in a resource file, you could use any of the following resource specifications:

`xclipboard.form`*`instance name for both the shell widget and form widget`*
`XClipboard.Form`*`class name for both the shell widget and form widget`*
`XClipboard.form`*`class name for the shell, and instance name for the form`*
`xclipboard.Form`*`instance name for the shell, and class name for the form`*

as well as any analogous loose bindings.

The `form` widget (named "parent" for internal reference within the application) is used in turn as the parent of the various command widgets and the text widget:

```
quit = XtCreateManagedWidget("quit", commandWidgetClass, parent, ... );
delete = XtCreateManagedWidget("delete", commandWidgetClass, parent,
... );
new = XtCreateManagedWidget("new", commandWidgetClass, parent, ... );
nextButton = XtCreateManagedWidget("next", commandWidgetClass, parent,
            ... );
prevButton = XtCreateManagedWidget("prev", commandWidgetClass, parent,
            ... );
text = XtCreateManagedWidget("text", textWidgetClass, parent, ... );
```

This "parent-child relationship" between composite widgets and their children is what is expressed in the instance hierarchy of the widget. So, for example, the Command widget instance named `quit` is a child of the Form widget instance named `form`, which in turn is a child of a Shell widget, which takes as its name the application name `xclipboard`.

G.3 What all this Means

The fully-specified instance name of any widget in an application is determined by the parent-child relationships of every widget in the application. First, there is always a shell widget, which takes as its name the application name. Then, there are one or more composite widgets, which contain other widgets. Finally, at the end of the chain, you have a simple widget, with the resources it defines, as well as the resources it inherits from its superclasses.

Don't confuse the class names of the widgets in the instance hierarchy with the class inheritance hierarchy of each widget. Figure G-4 tries to make the relationships clear.

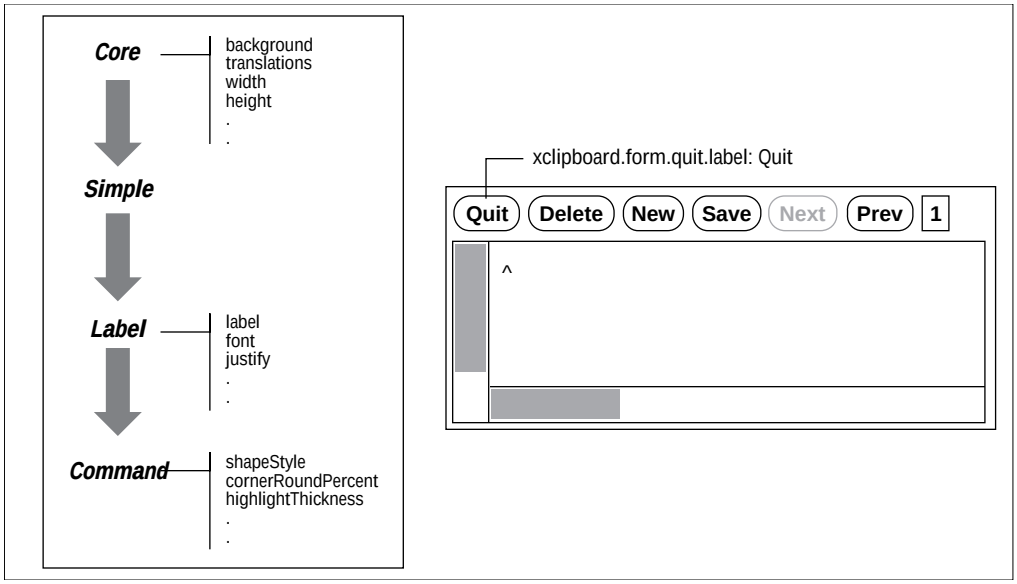


Figure G-4. Resource names and class inheritance

In Figure G-4, the `quit` widget gets its instance name from the relationship of widgets within the application. But it gets its resources from the class hierarchy of the widgets that the programmer used to develop the `Command` widget class.

Remember that the instance names of the widgets are completely arbitrary; even though it is not unusual to see a Form widget with the instance name `form`, there is nothing required about this. As a result, you need to look at the documentation for the application, not the widget, to find out the appropriate instance names.

The resources that a given widget class has are the result of its class inheritance hierarchy, which is defined by the widget programmer who originally designed the widget class. Thus, when you want to set resources for a widget like Command, you need to look not only in the section of this appendix that describes Command and its resources, but also the sections on each of its superclasses.

G.4 Complications

There are a number of provisos that modify this (hopefully by now clear and simple) picture:

- Even though a widget inherits a resource, it may not use it. For example, the Command widget class inherits the `borderWidth` resource from the Core widget class, but it does not actually use this information to redraw its border if you change the resource. A resource is just data you provide to the widget. Whether or not the widget does anything

with that data is up to its designer. If you set a resource and nothing seems to happen, you might have done something wrong...but you might also have set the resource correctly, and the widget simply chose to ignore it.

- Even when a widget does use a resource, you can't necessarily set it from a resource file. There are two reasons for this:
- The programmer has the option to "hardcode" the value of a resource when creating a widget. If he does this, all resource specifications for that resource are ignored.
- Some resources are designed only for programmer use. Some of these can't ever be specified in a resource file, since the data type of the resource isn't a text string, and the Toolkit doesn't provide any automatic conversion. (Things like colors or can be specified in resource files, even though a color name is not actually the color itself, because the X Toolkit automatically converts a color name to the appropriate internal format).

The following pages document only resources that are theoretically settable from resource files. (That is, if no converter exists, we've assumed that the resource is only for programmer use and have deleted it from the list.) However, there are many other resources listed that are most likely hardcoded by the programmer. Unfortunately, there is no way to tell in advance whether they will or will not be hardcoded in a particular application.

The "default values" listed for each widget resource may or may not apply to an actual application. These are the default values for the widget. An application can override them, either in the program code, or in an application defaults file. But inasmuch as the defaults are reasonable, they will usually be unchanged.

With this background, you're now ready to navigate the widget reference information contained in the next two appendices. For each widget (or gadget) in the OLIT and Athena widget sets, we give a brief description, a summary of its class hierarchy, a list of the new resources it defines, and its default translations and actions, if any.

Commercial X Toolkit-based applications should document all of the appropriate resources for the OLIT Motif widgets they use.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX H

OPEN LOOK Intrinsics Toolkit Widget Resources

As suggested on the reference pages for various clients, you can set not only resources defined by the application itself, but also resources that apply to any of the widgets that make up the application. The reference page for the application sometimes lists the most important of these resources, but for fuller customization, you need to know more about each widget.

Unfortunately, the design of the X Toolkit is such that to really do the right thing, you probably need to know a bit more about Toolkit programming than the average user might like.

In the previous appendix, we included some introductory concepts about how widgets are used in X Toolkit programs, and reference information about each class of Athena widgets. In this chapter we focus on the reference information needed to use programs based on the OLIT (OPEN LOOK Intrinsics Toolkit) toolkits used to produce many OPEN LOOK applications. The XView toolkit resources are covered in the next appendix. The final category of applications, NeWS applications under OpenWindows, do not make any use of X Resources.

H.1 Historical Note

The text of this Appendix was derived from AT&T-owned material for which copyright clearance was not available at press time. Since most users of this book are presumed to be using XView-based programs on Linux and SunOS, this will not be a serious problem. There is no implementation of OLIT for Linux, and SunOS users have the information available in online *man* pages and in the Answerbook.

H

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX I

Athena Widget Resources

As suggested on the reference pages for various clients, you can set not only resources defined by the application itself, but also resources that apply to any of the widgets that make up the application. The reference page for the application sometimes lists the most important of these resources, but for fuller customization, you need to know more about each widget.

Unfortunately, the design of the X Toolkit is such that to really do the right thing, you probably need to know a bit more about Toolkit programming than the average user might like. See Appendix G, *Introduction to Xt Widget Resources*, if you are not familiar with the gory details of "the Widget hierarchy."

In this appendix, we provide some reference information about each class of widgets in the Athena Widget set used by the standard MIT clients..

I.1 Box

The Box widget provides geometry management of arbitrary widgets in a box of a specified dimension. Box moves but does not resize its children. The children are rearranged when the Box is resized, when its children are resized, or when children are managed or unmanaged. The Box widget always attempts to pack its children as closely as possible within the geometry allowed by its parent.

Box widgets are commonly used to manage a related set of Command widgets and are frequently called ButtonBox widgets, but the children are not limited to buttons.

The children are arranged on a background that has its own specified dimensions and color.

I.1.1 Resources

The following new resources are associated with the Box widget:

`hSpace` (class `HSpace`) Number of pixels to the left or to the right of each child. Default is 4.

`orientation` (class `Orientation`) Specifies whether the preferred shape of the box is tall and narrow (`vertical`, the default) or short and wide (`horizontal`).

`vSpace` (class `VSpace`) Number of pixels above or below each child. Default is 4.

I.2 Command

The Command widget is an area, often rectangular, that contains a text or pixmap label and calls an application function when "pressed" with a pointer button. This selectable area is sometimes referred to as a "button." When the pointer cursor is on the button, the button border is highlighted to indicate that the button is ready for selection. When a pointer button is pressed, the command widget indicates that it has been selected by reversing its foreground and background colors.

I.2.1 Resources

The following new resources are associated with the Command widget:

`highlightThickness` (class `Thickness`) The thickness of the line drawn by the `highlight` action.

`shapeStyle` (class `ShapeStyle`) Nonrectangular buttons may be created using this resource. Nonrectangular buttons are supported only on a server that supports the Shape Extension. If nonrectangular buttons are specified for a server lacking this extension, the shape is ignored and the widgets will be rectangular. The following shape names are currently supported: `Rectangle`, `Oval`, `Ellipse`, and `roundedRectangle`, in any case.

`cornerRoundPercent` (class `CornerRoundPercent`) When a `ShapeStyle` of `roundedRectangle` is used, this resource controls the radius of the rounded corner. The radius of the rounded corners is specified as a percentage of the length of the shortest side of the widget.

I.2.2 Translations and Actions

The following are the default translation bindings that are used by the Command widget:

```
<EnterWindow>:highlight()
<LeaveWindow>:reset()
<Btn1Down>:set()
<Btn1Up>:notify() unset()
```

With these bindings, the user can cancel the action before releasing the button by moving the pointer out of the Command widget.

The full list of actions supported by Command is as follows:

`highlight()` Displays the internal highlight border in the color (`foreground` or `background`) that contrasts with the interior color of the Command widget. This is normally bound to <EnterWindow> events, so the widget border is highlighted when the pointer enters the window.

`unhighlight()` Displays the internal highlight border in the color (`foreground` or `background`) that matches the interior color of the Command widget. This action is called internally by reset(), and does not need an explicit translation.

`set()` Enters the `set` state, in which `notify` is possible and displays the interior of the button, including the highlight border, in the foreground color. The label is displayed in the background color. This usually happens when a pointer button (button 1 by default) is pressed in the widget.

`unset()` Cancels the `set` state and displays the interior of the button, including the highlight border, in the background color. The label is displayed in the foreground color. This action is called internally by reset(), and does not need an explicit translation.

`reset()` Cancels any `set` or `highlight` and displays the interior of the button in the background color, with the label displayed in the foreground color. This is normally bound to <Btu1Up> (along with `notify`()) to reset the button appearance to its normal state, once the button's callback function has been executed.

`notify()` Executes the callback list specified by `callback`, if executed in the `set` state. This is the action that actually calls the application function to be invoked.

I.3 Dialog

The Dialog widget prompts you for additional input. The typical Dialog widget contains three areas. The first line contains a description of the function of the Dialog widget, for example, the string *Filename:*. The second line contains an area into which you type input. The third line can contain buttons that let you confirm or cancel the Dialog input.

Dialog is not really a widget, but an interface to a widget. It might also be thought of as a compound widget. It includes a label widget, a command widget, and a text widget as components. These could theoretically appear as subwidgets in a resource specification.

I.3.1 Resources

The following new resources are associated with the Dialog widget:

`icon` (class `Icon`) The name of a pixmap to be displayed immediately to the left of the Dialog widget's label.

`label` (class `Label`) A string to be displayed at the top of the Dialog widget.

`value` (class `Value`) An initial value for the string field into which you will enter text. By default, no text entry field is available. Specifying an initial value for `value` activates the text entry field. If string input is desired but no initial value is to be specified, then set this resource to `""` (empty string).

I.4 Form

The Form widget can contain an arbitrary number of children of any class. The Form provides geometry management for its children, including individual control of the position of each child. The initial positions of the children may be computed relative to the positions

of other children. When the Form is resized, it computes new positions and sizes for its children.

I.4.1 Resources

The following new resource is associated with the Form widget:

`defaultDistance` Specifies the default value for `horizDistance` and `vertDistance`. This value is four pixels, by default. The default width of the Form is the minimum width needed to enclose the children after computing their initial layout, with a margin of `defaultDistance` at the right and bottom edges. If a width and height is assigned to the Form that is too small for the layout, the children will be clipped by the right and bottom edges of the Form.

Form is a subclass of Constraint, which means it has special kind of resources called constraint resources, which means that they apply to—and are specified as if they belong to—the child of the Form rather than to the Form itself. For example, *xcalc* uses a Form widget to organize its buttons. The resources below apply to the buttons, rather than to the Forms (e.g., xcalc.ti.button11.horizDistance : 4)

`bottom` (class `Edge`)

`top` (class `Edge`)

`left` (class `Edge`)

`right` (class `Edge`) Specify how to reposition the bottom, top, left, and right, respectively, of a child widget when the Form is resized. These resources can take one of five values. The values `ChainTop`, `ChainBottom`, `ChainLeft`, and `ChainRight` maintain a constant distance from an edge of the child to the top, bottom, left, and right edges, respectively, of the Form. The value `Rubber` (default) maintains a proportional distance from the edge of the child to the left or top edge of the Form. The proportion is determined from the initial position of the child and the initial size of the Form.

`fromHoriz` (class `Widget`)

`horizDistance` (class `Thickness`) Specify a child widget's horizontal position relative to another widget within the Form. `fromHoriz` is the name of the widget relative to which the child widget is placed, and `horizDistance` is the number of pixels separating the two widgets. For example, if `horizDistance` is 10, the child widget will be placed 10 pixels to the right of the widget defined in `fromHoriz`. If `fromHoriz` is not defined, then `horizDistance` is measured from the left edge of the Form.

`fromVert` (class `Widget`)

`vertDistance` (class `Thickness`) Similar to previous resources, except that `fromVert` and `vertDistance` position a child widget by a specified number of pixels *vertically* away from a specified widget. If no widget is specified for `fromVert`, then `vertDistance` is measured from the top of the Form.

`resizable` (class `Boolean`) `TRUE` if children are allowed to resize themselves. Default is `FALSE`.

I.5 Grip

The Grip widget provides a small region that allows button presses and button releases. The Grip widget is typically used as an attachment point for visually repositioning an object (for example, the pane border in a Paned widget).

I.5.1 Resources

The following Core resources may be useful with the Grip widget:

`foreground, width, height, borderWidth`.

I.5.2 Translations and Actions

The Grip widget does not declare any default event translation bindings, but it does declare a single action routine named `GripAction` in its action table. The client specifies an arbitrary event translation table, giving parameters to the `GripAction` routine.

The following is an example of a `GripAction` translation table:

```
<Btn1Down>: GripAction(press)
<Btn1Motion>: GripAction(move)
<Btn1Up>: GripAction(release)
```

I.6 Label

A Label is an non-editable text string or pixmap that is displayed within a window. The string may contain multiple lines of characters. It can be aligned to the left, right, or center of its window. A Label can be neither selected nor directly edited by the user.

I.6.1 Resources

The following resources are used by the Label widget:

`bitmap` (class `Pixmap`) Specifies a bitmap to display in place of the text label. In a resource file, the resource should be specified as the name of a file in the bitmap utility format that is to be loaded into a pixmap. The string can be an absolute or a relative filename. If a relative filename is used, the directory specified by the resource name `bitmapFilePath` or the resource class `BitmapFilePath` is added to the beginning of the specified filename. If the `bitmapFilePath` resource is not defined, the default directory on a POSIX-based system is */usr/include/X11/bitmaps* .

`font` (class `font`) The font of the label.

`foreground` (class `Foreground`) The color of the text string or pixmap.

`internalHeight` (class `Height`) Represents the distance in pixels between the top and bottom of the label text or bitmap and the horizontal edges of the Label widget. Default is 2 pixels.

`internalWidth` (class `Width`) Represents the distance in pixels between the ends of the label text or bitmap and the vertical edges of the Label widget. Default is 4 pixels.

`justify` (class `Justify`) Specifies left, center, or right alignment of the label string within the Label widget. One of the values `left`, `center`, or `right` can be specified.

`label` (class `Label`) Specifies the text string that is to be displayed in the button if no bitmap is specified. Note that the label may be hardcoded by the application.

`resize` (class `Resize`) A Boolean value that specifies whether the Label widget should attempt to resize to its preferred dimensions whenever `XtSetValues` is called for it. Default is `True`. Not usually set by users.

`rowSpacing` (class `Spacing`) Specify the amount of space in pixels between each of the rows in the list. The default is 6 pixels.

I.7 List

The List widget is a rectangle that contains a list of text strings formatted into rows and columns. When one of the strings is selected, it is highlighted, and an application callback routine is invoked. Only one string may be selected at a time. Note that most of the List resources are for application use.

I.7.1 Resources

The following resources are used by the List widget:

`callback` All functions on this list are called whenever the `notify` action is invoked. This resource cannot be set from within a resource file, but only within an application program.

`columnSpacing` (class `Spacing`) Specify the amount of space in pixels between each of the columns in the list. The default is 6 pixels.

`Cursor` (class `Cursor`) The cursor to be displaced when the pointer is in the List widget. Default is `left-ptr`.

`defaultColumns` (class `Columns`) Specifies the default number of columns, which is used when neither the width nor the height of the List widget is specified or when `forceColumns` is `True`. The default is 2.

`forceColumns` (class `Columns`) Specifies that the default number of columns is to be used no matter what the current size of the List widget is. The default is FALSE.

`font` (class `Font`) Specifies the font to be used to display the list.

`foreground` (class `Foreground`) Specifies the color to be used to paint the text of the list elements.

`internalHeight` (class `Height`) Represents a margin, in pixels, between the top and bottom of the list and the edges of the List widget. Default is 2 pixels.

`internalWidth` (class `Width`) Represents a margin, in pixels, between the left and right edges of the list and the edges of the List widget. Default is 4 pixels.

`longest` (class `Longest`) Specifies the length, in pixels, of the longest string in the current list. If the client knows the length, it should specify it; otherwise, the List

widget computes a default length by searching through the list. This value is not typically set in resource files.

`numberStrings` (class `NumberStrings`) Specifies the number of strings in the current list. If a value is not specified, the list must be `NULL`-terminated. This value is not typically set in resource files.

`pasteBuffer` (class `Boolean`) If this is `True`, then the value of the string selected will be put into X cut buffer 0. The default is FALSE. (Normally, the selected item is simply passed to the application. For example, a filename might be passed to the application's "open" routine.)

`verticalList` (class `Boolean`) If this is `True`, the elements in the list are arranged vertically; if `False`, the elements are arranged horizontally.

I.7.2 Translations and Actions

The List widget has three predefined actions: `Set`, `Unset`, and `Notify`. `Set` and `Unset` allow switching the foreground and background colors for the current list item. `Notify` allows processing application callbacks.

The following is the default translation table used by the List widget:

```
<Btn1Down>,<Btn1Up>:Set() Notify()
```

These translations should not typically be modified by users, and may be hardcoded by the application.

I.8 MenuButton

The MenuButton widget is a subclass of the Command widget that is used to pop-up a menu. It is an area, often rectangular, that contains a text or pixmap label. This selectable area is referred to as a button. When the pointer cursor is on the button, the button border is highlighted to indicate that the button is ready for selection. When pointer button 1 is pressed, the MenuButton widget pops up the menu that has been named in the `menuName` resource.

I.8.1 Resources

MenuButton widgets have no new user-settable resources.

I.8.2 Translations and Actions

The following default translation bindings are used by the MenuButton widget:

```
<EnterWindow>:highlight()
    <LeaveWindow>:reset()
    <BtnDown>:reset(|) PopupMenu()
```

With these bindings, the user can cancel the action before releasing the button by moving the pointer out of the MenuButton widget.

The actions supported by MenuButton are listed below:

`highlight(`***`condition`***`)` Displays the internal highlight border in the color (`foreground` or `background`) that contrasts with the interior color of the MenuButton widget. The conditions `WhenUnset` and `Always` are understood by this action procedure. If no argument is passed, `WhenUnset` is assumed.

`unhighlight()` Displays the internal highlight border in the color (`foreground` or `background`) that matches the interior color of the MenuButton widget.

`set()` Enters the set state, in which `notify` is possible, and displays the interior of the button, including the highlight border, in the foreground color. The label or pixmap is displayed in the background color.

`unset()` Cancels the set state and displays the interior of the button, including the highlight border, in the background color. The label or pixmap is displayed in the foreground color.

`reset()` Cancels any `set` or `highlight` action and displays the interior of the button in the background color, with the label or pixmap displayed in the foreground color.

`PopupMenu()` Pops up the menu specified by the `menuName` resource.

I.9 Paned

The Paned widget manages children in a vertically or horizontally tiled fashion. You may resize these panes by using the *grips* that appear near the right or bottom edge of the border between two panes.

When you position the pointer on a grip, pressing the pointer button will display an arrow that indicates which pane is being resized. By keeping the pointer button down, you can move the pointer up and down (or left and right). This, in turn, changes the border between the panes, causing one pane to shrink and some other pane (or panes) to grow. The size of the Paned widget will not change.

The choice of alternate pane is a function of the `min`, `max`, and `skipAdjust` constraints on the other panes. With the default bindings, button 1 resizes the pane above or to the left of the selected grip, button 3 resizes the pane below or to the right of the selected grip, and button 2 repositions the border between two panes only.

I.9.1 Resources

The following new resources are associated with the Paned widget:

`betweenCursor` (class `Cursor`) Cursor for changing the boundary between two panes.

`cursor` (class `Cursor`) Pointer cursor image that displays, whenever the pointer is in this widget but not in any of its children (children may also inherit this cursor).

`gripCursor` (class `Cursor`) Cursor for grip when not active.

`gripIndent` (class `GripIndent`) Offset of grip from margin (in pixels). Default is 10.

`gripTranslations` (class `Translations`) Button bindings for grip.

`horizontalBetweenCursor` (class `Cursor`) Cursor to use for the grip when changing the boundary between two panes. Default is `sb_up_arrow`.

`horizontalGripCursor` (class `Cursor`) Cursor to use for the grips when they are not active. Default is `sb_h_double_arrow`.

`internalBorderColor` (class `BorderColor`) Internal border color of the widget's window. Default is `XtDefaultForeground`.

`internalBorderWidth` (class `BorderWidth`) Amount of space (in pixels) kept between panes. Default is 1.

`leftCursor` (class `Cursor`) Cursor used when resizing the pane to the left of the grip. Default is `sb_left_arrow`.

`lowerCursor` (class `Cursor`) Cursor used when resizing the pane below the grip. Default is `sb_down_arrow`.

`orientation` (class `Orientation`) Orientation to use in stacking the panes. This value can be either `Vertical` (the default) or `Horizontal`.

`refigureMode` (class `Boolean`) A Boolean that specifies whether the Paned widget should adjust its children. Default is `TRUE`.

`rightCursor` (class `Cursor`) Cursor used when resizing the pane to the right of the grip. Default is `sb_right_arrow`.

`upperCursor` (class `Cursor`) Cursor used when resizing the pane above the grip. Default is `sb_up_arrow`.

`verticalBetweenCursor` (class `Cursor`) Cursor to use for the grip when changing the boundary between two panes. Default is `sb_left_arrow`.

`verticalGripCursor` (class `Cursor`) Cursor to use for the grips when they are not active. Default is `sb_v_double_arrow`.

Paned supports the following constraint resources. They can be specified to the Paned widget to indicate where a child widget should be positioned within the Paned widget.

`allowResize` (class `Boolean`) A Boolean that specifies whether to accept a child's request to resize. The default, `FALSE`, is to ignore such requests.

`max` (class `max`) Maximum height for pane. Default is to allow unlimited height.

`min` (class `min`) Minimum height for pane (in pixels). Default is 1.

`preferredPaneSize` (class `PreferredPaneSize`) Preferred size of pane. This default is dependent on the application.

`resizeToPreferred` (class `Boolean`) A Boolean that specifies whether to resize each pane to its preferred size when the Paned widget is resized. Default is `False`.

`showGrip` (class `ShowGrip`) A Boolean that specifies whether to show a grip for this pane. Default is `True`.

`skipAdjust` (class `Boolean`) By default, this resource is `FALSE`, meaning that the Paned widget will resize a pane automatically, whenever necessary. If this

resource is TRUE, the Paned widget will skip the adjustment of the pane. .SH "Translations and Actions" The Paned widget has no action routines of its own, as all actions are handled through the grips. The grips are each assigned a default Translation table.

```
<Btn1Down>:GripAction(Start, UpLeftPane)
    <Btn2Down>:GripAction(Start, ThisBorderOnly)
    <Btn3Down>:GripAction(Start, LowRightPane)
    <Btn1Motion>:GripAction(Move, UpLeftPane)
    <Btn2Motion>:GripAction(Move, ThisBorderOnly)
    <Btn3Motion>:GripAction(Move, LowRightPane)
    Any<BtnUp>:GripAction(Commit)
```

The Paned widget interprets the GripAction as taking two arguments. The first argument may be any of the following:

```
StartUpLeftPane, ThisBorderOnly, LowRightPane).
```

MoveStart action that began this process. If these arguments are not passed, the behavior is undefined.

Commit

I.10 Scrollbar

The Scrollbar widget is a rectangular area that contains a slide region and a thumb (slide bar). A Scrollbar can be used alone (to provide a graduated scale) or within a composite widget (for example, a Viewport). A Scrollbar can be aligned either vertically or horizontally.

When a Scrollbar is created, it is drawn with the thumb in a contrasting color. The thumb is normally used to scroll client data and to give visual feedback on the percentage of the client data that is visible.

I.10.1 Resources

You can set the dimensions of the Scrollbar two ways:

- By using the width and height resources, as you can for all widgets.
- By using the Scrollbar resources length and thickness, which are independent of the vertical or horizontal orientation.

The following new resources are associated with the Scrollbar widget:

foreground (class Foreground) Thumb color.

length (class Length) Specifies the height for a vertical Scrollbar and the width for a horizontal Scrollbar. Default is 1 (pixel).

minimumThumb (class MinimumThumb) Smallest size, in pixels, to which the thumb can shrink. Default is 7.

orientation (class Orientation) Orientation of scrollbar. This value can be either XtOrientVertical (the default) or XtOrientHorizontal. Not usually set in resource files.

`scrollDCursor` (class `Cursor`) Cursor for scrolling down. Default is `XC_sb_down_arrow`.

`scrollHCursor` (class `Cursor`) Idle horizontal cursor. Default is `XC_sb_h_double_arrow`.

`scrollLCursor` (class `Cursor`) Cursor for scrolling left. Default is `XC_sb_left_arrow`.

`scrollRCursor` (class `Cursor`) Cursor for scrolling right. Default is `XC_sb_right_arrow`.

`scrollUCursor` (class `Cursor`) Cursor for scrolling up. Default is `XC_sb_up_arrow`.

`scrollVCursor` (class `Cursor`) Idle vertical cursor. Default is `XC_sb_v_double_arrow`.

`shown` (class `Shown`) Percentage the thumb covers. Default is 0.0.

`thickness` (class `Thickness`) Specifies the width for a vertical Scrollbar and the height for a horizontal Scrollbar. Default is 14 (pixels).

`thumb` (class `Thumb`) Thumb pixmap. Default is `GrayPixmap`.

`topOfThumb` (class `TopOfThumb`) Position on scroll bar. Default is 0.0.

I.10.2 Translations and Actions

The actions supported by the Scrollbar widget are:

`StartScroll(`***`value`***`)` The possible values are `Forward`, `Backward`, or `Continuous`. This must be the first action to begin a new movement.

`NotifyScroll(`***`value`***`)` The possible values are `Proportional` or `FullLength`. If the argument to `StartScroll` was `Forward` or `Backward`, `NotifyScroll` executes the `XtNscrollProc` callbacks and passes either the position of the pointer if its argument is `Proportional` or the full length of the scroll bar if its argument is `FullLength`. If the argument to `StartScroll` was `Continuous`, `NotifyScroll` returns without executing any callbacks.

`EndScroll()` This must be the last action after a movement is complete.

`MoveThumb()` Repositions the scroll bar thumb to the current pointer location.

`NotifyThumb()` Calls the `XtNjumpProc` callbacks and passes the relative position of the pointer as a percentage of the scroll bar length.

The default bindings for Scrollbar are:

```
<Btn1Down>:StartScroll(Forward)
    <Btn2Down>:StartScroll(Continuous) MoveThumb() NotifyThumb()
    <Btn3Down>:StartScroll(Backward)
    <Btn2Motion>:MoveThumb() NotifyThumb()
    <BtnUp>:NotifyScroll(Proportional) EndScroll()
```

Examples of additional bindings you might wish to specify in a resource file are:

```
*Scrollbar.Translations: \
~<KeyPress>f: StartScroll(Forward) NotifyScroll(FullLength) EndScroll()
```

```
<KeyPress>b: StartScroll(Backward) NotifyScroll(FullLength) EndScroll()
```

I.11 Simple

The Simple widget defines characteristics that are inherited by non-composite widgets such as Labels, Lists, and Scrollbars. The Simple widget never appears in applications, but it does define resources that are inherited by its subclasses.

I.11.1 Resources

The following resources are associated with the Simple widget:

`cursor` (class `Cursor`) The cursor to use within the widget. Default is none.

`insensitiveBorder` (class `Insensitive`) The pixmap to use to indicate that the Simple widget cannot receive input. Default is `Gray`.

I.12 SimpleMenu

The SimpleMenu widget is a container for menu entries. It is a direct subclass of Shell. This is the only part of the menu that actually contains a window, since each menu pane is a gadget (a widget without a window). SimpleMenu "glues" the individual menu entries together into one menu.

I.12.1 Resources

The following new resources are used by the SimpleMenu widget:

`backingStore` (class `BackingStore`) Determines what type of backing store will be used for the menu. Legal values for this resource are `NotUseful`, `WhenMapped`, and `Always`. These values are the backing-store integers defined in *<X11/X.h>*. If `default` is specified (the default behavior) the server will use whatever it thinks is appropriate. This resource is typically set by the application.

`bottomMargin`, `topMargin` (class `VerticalMargins`) The amount of space between the top or bottom of the menu and the menu entry closest to that edge. Default is 0.

`cursor` (class `Cursor`) The shape of the mouse pointer whenever it is in this widget.

`popupOnEntry` (class `PopupOnEntry`) The `XawPositionSimpleMenu` action pops up the SimpleMenu with its label (or first entry) directly under the pointer, by default. To pop up the menu under another entry, the application can set this resource to the menu entry that *should* be under the pointer when the menu is popped up. This allows the application to offer the user a default menu entry that can be selected without moving the pointer. Not usually settable by the user.

`rowHeight` (class `RowHeight`) If this resource is 0 (the default), then each menu entry is given its desired height. If this resource has any other value, then all menu entries are forced to be `rowHeight` pixels high.

I.12.2 Translations and Actions

The following default translation bindings are used by the SimpleMenu widget:

```
<EnterWindow>:highlight()
    <LeaveWindow>:unhighlight()
    <BtnMotion>:highlight()
    <BtnUp>:MenuPopdown() notify() unhighlight()
```

With these bindings, the user can pop down the menu without activating any of the callback functions, by releasing the pointer button when no menu item is highlighted.

The actions supported by SimpleMenu are listed below:

`highlight()` Highlights the menu entry that is currently under the pointer. Only an item that is highlighted is notified when the `notify` action is invoked. The look of a highlighted entry is determined by the menu entry.

`unhighlight()` Unhighlights the currently highlighted menu item and returns it to its normal look.

`notify()` Notifies the currently highlighted menu entry that it has been selected. It is the responsibility of the menu entry to take the appropriate action.

`MenuPopdown(`***menu***`)` Built-in action to pop down a menu widget.

I.13 Sme

The Sme object is the base class for all menu entries that are children of SimpleMenu. While this object is intended mainly to be subclassed, it may be used in a menu to add blank space between menu entries.

I.13.1 Resources

The Sme object defines no new resources.

I.14 SmeBSB

The SmeBSB object is used to create a menu entry that contains a string, and optional bitmaps in its left and right margins. The parent is expected to be SimpleMenu. Since each menu entry is an independent object, the application is able to change the font, color, height, and other attributes of the menu entries, on an entry-by-entry basis.

I.14.1 Resources

The following resources are used by the SmeBSB object:

`font` (class `Font`) Specifies the font used by the menu entry.

`foreground` (class `Foreground`) Specifies the foreground color of the menu entry's window. This color is also used to render all 1's in `leftBitmap` and `rightBitmap`.

`justify` (class `Justify`) Specifies how the label is to be rendered between the left and right margins when the space is wider than the actual text. When specifying the

justification from a resource file, the values `left`, `center`, or `right` may be used.

`label` (class `Label`) Specifies the string to be displayed in the menu entry. The exact location of this string within the bounds of the menu entry is controlled by the resources `leftMargin`, `rightMargin`, `vertSpace`, and `justify`.

`leftBitmap` (class `LeftBitmap`), `rightBitmap` (class `RightBitmap`) Specifies a name of a bitmap to display in the left or right margin of the menu entry. All 1's in the bitmap are rendered in the foreground color of the SimpleMenu widget, and all 0's will be drawn in the background color of the SimpleMenu widget. The programmer must ensure that the menu entry is tall enough and that the appropriate margin is wide enough to accept the bitmap. If care is not taken, the bitmap might extend into either another menu entry or this entry's label. This resource is typically set by the application.

`leftMargin`, `rightMargin` (class `HorizontalMargins`) Specifies the amount of space (in pixels) to leave between the edge of the menu entry and the label string.

`vertSpace` (class `VertSpace`) Specifies the amount of vertical padding to place around the label of a menu entry. The label and bitmaps are always centered vertically within the menu. Values for this resource are expressed as a percentage of the font's height. The default value (25) increases the default height to 125% of the font's height.

I.15 SmeLine

The SmeLine object is used to add a horizontal line or menu separator to a SimpleMenu. Since each menu entry is an independent object, the application is able to change the color, height, and other attributes of the menu entries, on an entry-by-entry basis. This entry is not selectable, and does not highlight when the pointer cursor is over it.

I.15.1 Resources

The following resources are used by the SmeLine object:

`foreground` (class `Foreground`) The foreground color of the menu entry's window.

`lineWidth` (class `LineWidth`) The width of the horizontal line to be displayed.

`stipple` (class `Stipple`) If a bitmap is specified for this resource, the line will be stippled through it. This allows the menu separator to be rendered as something more exciting than just a line. For instance, if the application defines a stipple that is a chain link, then menu separators will look like chains.

I.16 StripChart

The StripChart widget is used to provide a real-time graphic chart of a single value. This widget is used by *xload* to provide the load graph. It will read data from an application, and update the chart at the interval specified by `update`.

I.16.1 Resources

The following resources are used by the StripChart widget:

`height` (class `Height`) The height of the stripchart. Default is 120 pixels.

`highlight` (class `Foreground`) The color that will be used to draw the scale lines on the graph.

`jumpScroll` (class `JumpScroll`) When the graph reaches the right edge of the window it must be scrolled to the left. This resource specifies the number of pixels it will jump. Smooth scrolling can be achieved by setting this resource to 1.

`minScale` (class `Scale`) The minimum scale for the graph. The number of divisions on the graph will always be greater than or equal to this value. Default is 1.

`update` (class `Interval`) The number of seconds between graph updates. Each update is represented on the graph as a 1-pixel-wide line. Every `update` seconds, a new graph point will be added to the right end of the StripChart. Default is 10.

`width` (class `Width`) The width of the stripchart. Default is 120 pixels.

I.17 Text

A Text widget is a window that provides a way for an application to display one or more lines of text. The displayed text can reside in a file on disk or in a string in memory. An option also lets an application display a vertical Scrollbar in the Text window, letting the user scroll through the displayed text. Other options allow an application to let the user modify the text in the window or search for a specific string.

Three types of edit mode are available: .RS 3n

" Append-only mode lets the user enter text into the window, while read-only mode does not. Text may be entered only if the insertion point is after the last character in the window. Editable mode lets you place the cursor anywhere in the text and modify the text at that position. The text cursor position can be modified by using the keystrokes or pointer buttons defined by the event bindings. (See the section "Translations and Actions" below.)

I.17.1 Resources

The following resources are used by the Text widget:

`autoFill` (class `AutoFill`) A Boolean that specifies whether the Text widget will automatically break a line when the user attempts to type into the right margin Default is `False`.

`bottomMargin` (class `Margin`) Amount of space, in pixels, between the edge of the window and the edge of the text within the window. Default is 2.

`dataCompression` If `True` (the default), the `AsciiSrc` will compress its data to the minimum size required. This will happen either every time the text string is saved or whenever the value of the string is queried.

`displayCaret` (class `Output`) A Boolean that specifies whether to display the text caret. Default is `True`.

`displayPosition` (class `TextPosition`) Character position of first line. Default is 0.

`insertPosition` (class `TextPosition`) Character position of caret. Default is 0.

`leftMargin` (class `Margin`) Left margin in pixels. Default is 2.

`rightMargin` (class `Margin`) Amount of space, in pixels, between the edge of the window and the corresponding edge of the text within the window. Default is 2.

`scrollHorizontal`

`scrollVertical` Control the placement of scrollbars on the left and bottom edge of the text widget. Possible values are `XawtextScrollAlways`, `XawtextScrollWhenNeeded`, and `XawtextScrollNever` (the default). Not settable from a resource file.

`topMargin` (class `Margin`) Amount of space, in pixels, between the edge of the window and the corresponding edge of the text within the window. Default is 2.

`useStringInPlace` If `True`, will disable the memory management provided by the Text widget, updating the `string` resource instead. Default is `False`.

I.17.2 Translations and Actions

Many standard keyboard editing facilities are supported by the event bindings. The following actions are supported: l| l. Cursor MovementDelete

–

.TH Forward-characterDelete-next-character Backward-characterDelete-previous-character Forward-wordDelete-next-word Backward-wordDelete-previous-word Forward-paragraphDelete-selection Backward-paragraph Beginning-of-lineSelection End-of-line Insert-selection Next-lineSelect-word Previous-lineSelect-all Next-pageSelect-start Previous-pageSelect-adjust Beginning-of-fileSelect-end End-of-fileExtend-start Scroll-one-line-upExtend-adjust Scroll-one-line-downExtend-end New LineMiscellaneous Newline-and-indentRedraw-display Newline-and-backupInsert-file NewlineInsert-char KillDisplay-caret Kill-wordFocus-in Backward-kill-wordFocus-out Kill-selectionSearch Kill-to-end-of-lineMultiply Kill-paragraphForm-paragraph Kill-to-end-of-paragraphTranspose-characters No-op

–

$delete action deletes a text item. The `kill` action deletes a text item and puts the item in the kill buffer (X cut buffer 1).

$insert-selection action retrieves the value of a specified X selection or cut buffer, with fallback to alternative selections or cut buffers.

I.17.2.1 Cursor Movement Actions

`forward-character()`

`backward-character()` These actions move the insert point forward or backward one character in the buffer. If the insert point is at the end (or beginning) of a line, this action moves the insert point to the next (or previous) line.

`forward-word()`

`backward-word()` These actions move the insert point to the next or previous word boundary. A word boundary is defined as a space, a tab, or a carriage return.

`forward-paragraph()`

`backward-paragraph()` These actions move the insert point to the next or previous paragraph boundary. A paragraph boundary is defined as two carriage returns in a row with only spaces or tabs between them.

`beginning-of-line()`

`end-of-line()` These actions move to the beginning or end of the current line. If the insert point is already at the end or beginning of the line, no action is taken.

`next-line()`

`previous-line()` These actions move the insert point up or down one line. If the insert point is currently *n* characters from the beginning of the line then it will be *n* characters from the beginning of the next or previous line. If *n* is past the end of the line, the insert point is placed at the end of the line.

`next-page()`

`previous-page()` These actions move the insert point up or down one page in the file. One page is defined as the current height of the text widget. These actions always place the insert point at the first character of the top line.

`beginning-of-file()`

`end-of-file()` These actions place the insert point at the beginning or end of the current text buffer. The text widget is then scrolled the minimum amount necessary to make the new insert point location visible.

`scroll-one-line-up()`

`scroll-one-line-down()` These actions scroll the current text field up or down by one line. They do not move the insert point. Other than the scrollbars, this is the only way that the insert point may be moved off of the visible text area. The widget will be scrolled so that the insert point is back on the screen as soon as some other action is executed.

I.17.2.2 Delete Actions

`delete-next-character()`

`delete-previous-character()` These actions remove the character immediately after or before the insert point. If a carriage return is removed, the next line is appended to the end of the current line.

`delete-next-word()`

`delete-previous-word()` These actions remove all characters between the insert point location and the next word boundary. A word boundary is defined as a space, a tab or a carriage return.

`delete-selection()` This action removes all characters in the current selection. The selection can be set with the selection actions.

I.17.2.3 Selection Actions

`select-word()` This action selects the word in which the insert point is currently located. If the insert point is between words, it will select the previous word.

`select-all()` This action selects the entire text buffer.

`select-start()` This action sets the insert point to the current pointer location, where a selection then begins. If many of these selection actions occur quickly in succession then the selection count mechanism will be invoked.

`select-adjust()` This action allows a selection started with the `select-start` action to be modified, as described above.

`select-end(`***name***[,***name***,...]) This action ends a text selection that began with the `select-start` action, and asserts ownership of the selection or selections specified. A ***name*** can be a selection (e.g., PRIMARY) or a cut buffer (e.g., CUT_BUFFER0). Note that case is important. If no ***names*** are specified, PRIMARY is asserted.

`extend-start()` This action finds the nearest end of the current selection, and moves it to the current pointer location.

`extend-adjust()` This action allows a selection started with an `extend-start` action to be modified.

`extend-end(`***name***[,***name***,...]) This action ends a text selection that began with the `extend-start` action, and asserts ownership of the selection or selections specified. A ***name*** can be a selection (e.g., PRIMARY) or a cut buffer (e.g., CUT_BUFFER0). Note that case is important. If no ***name*** is given, PRIMARY is asserted.

`insert-selection(`***name***[,***name***,...]) This action retrieves the value of the first (leftmost) named selection that exists or the cut buffer that is not empty. This action then inserts it into the Text widget at the current insert point location. A ***name*** can be a selection (e.g., PRIMARY) or a cut buffer (e.g., CUT_BUFFER0). Note that case is important.

I.17.2.4 New Line Actions

`newline-and-indent()` This action inserts a newline into the text and adds spaces to that line to indent it to match the previous line. (Note: this action still has a few bugs.)

`newline-and-backup()` This action inserts a newline into the text *after* the insert point.

`newline()` This action inserts a newline into the text *before* the insert point.

I.17.2.5 Kill Actions

`kill-word()`

`backward-kill-word()` These actions act exactly like the `delete-next-word` and `delete-previous-word` actions, but they store the word that was killed into the kill buffer (CUT_BUFFER_1).

`kill-selection()` This action deletes the current selection and stores the deleted text into the kill buffer (CUT_BUFFER_1).

`kill-to-end-of-line()` This action deletes the entire line to the right of the insert point, and stores the deleted text into the kill buffer (CUT_BUFFER_1).

`kill-paragraph()` This action deletes the current paragraph. If the insert point is between paragraphs, it deletes the paragraph above the insert point, and stores the deleted text into the kill buffer (CUT_BUFFER_1).

`kill-to-end-of-paragraph()` This action deletes everything between the current insert point and the next paragraph boundary, and puts the deleted text into the kill buffer (CUT_BUFFER_1).

I.17.2.6 Miscellaneous Actions

`redraw-display()` This action recomputes the location of all the text lines on the display, scrolls the text to center vertically the line containing the insert point on the screen, clears the entire screen, and then redisplays it.

`insert-file([`***`filename`***`)` This action activates the insert file popup. The ***`filename`*** option specifies the default filename to put in the filename buffer of the popup. If no ***`filename`*** is specified the buffer is empty at startup.

`insert-char()` This action may be attached only to a key event. It calls `XLookupString` to translate the event into a (rebindable) Latin-1 character (sequence) and inserts that sequence into the text at the insert point position.

`insert-string(`***`string`***`[,`***`string`***`,...])` This action inserts each ***`string`*** into the text at the insert point location. Any ***`string`*** beginning with the characters "0x" and containing only valid hexadecimal digits in the remainder is interpreted as a hexadecimal constant and the corresponding single character is inserted instead.

`display-caret(`***`state`***`,`***`when`***`)` This action allows the insert point to be turned on and off. The ***`state`*** argument specifies the desired state of the insert point. This value may be any of the string values accepted for Boolean resources (e.g., `on`, `True`, `off`, `False`, etc.). If no arguments are specified, the default value is `True`. The ***`when`*** argument specifies, for `EnterNotify` or `LeaveNotify` events, whether or not the focus field in the event is to be examined. If the second argument is not specified, or specified as something other than `always`, then if the

action is bound to an `EnterNotify` or `LeaveNotify` event, the action will be taken only if the focus field is `True`. An augmented binding that might be useful is:

```
*Text.Translations: #override \
<FocusIn>:display-caret(on) \n\
<FocusOut>:display-caret(off)
```

`focus-in()`

`focus-out()` These actions do not currently do anything.

`search(`***direction***`,[`***string***`)` This action activates the search popup. The ***direction*** must be specified as either `forward` or `backward`. The string is optional and is used as an initial value for the "Search for:" string.

`multiply(`***value***`)` The multiply action allows the user to multiply the effects of many of the text actions. Thus the following action sequence:

```
multiply(10) delete-next-word()
```

will delete 10 words. It does not matter whether these actions take place in one event or many events. Using the default translations the key sequence Control-u, Control-d will delete 4 characters. Multiply actions can be chained; thus,

```
multiply(5) multiply(5)
```

is the same as:

```
multiply(25)
```

If the string `reset` is passed to the multiply action the effects of all previous multiplies are removed and a beep is sent to the display.

`form-paragraph()` This action removes all the carriage returns from the current paragraph and reinserts them so that each line is as long as possible, while still fitting on the current screen. Lines are broken at word boundaries if at all possible. This action currently works only on Text widgets that use ASCII text.

`transpose-characters()` This action will switch the positions of the character to the left of the insert point and the character to the right of the insert point. The insert point will then be advanced one character.

`no-op([`***action***`)` The no-op action makes no change to the text widget, and is used mainly to override translations. This action takes one optional argument. If this argument is `RingBell` then a beep is sent to the display.

I.17.2.7 Event Bindings

The default event bindings for the Text widget are:

```
char defaultTextTranslations[] = "\ Ctrl<Key>F:forward-character() \n\
            Ctrl<Key>B:backward-character() \n\
            Ctrl<Key>D:delete-next-character() \n\
            Ctrl<Key>A:beginning-of-line() \n\
            Ctrl<Key>E:end-of-line() \n\
            Ctrl<Key>H:delete-previous-character() \n\
            Ctrl<Key>J:newline-and-indent() \n\
            Ctrl<Key>K:kill-to-end-of-line() \n\
            Ctrl<Key>L:redraw-display() \n\
            Ctrl<Key>M:newline() \n\
            Ctrl<Key>N:next-line() \n\
            Ctrl<Key>O:newline-and-backup() \n\
            Ctrl<Key>P:previous-line() \n\
            Ctrl<Key>V:next-page() \n\
            Ctrl<Key>W:kill-selection() \n\
            Ctrl<Key>Y:unkill() \n\
            Ctrl<Key>Z:scroll-one-line-up() \n\
            Meta<Key>F:forward-word() \n\
            Meta<Key>B:backward-word() \n\
            Meta<Key>I:insert-file() \n\
            Meta<Key>K:kill-to-end-of-paragraph() \n\
            Meta<Key>V:previous-page() \n\
            Meta<Key>Y:stuff() \n\
            Meta<Key>Z:scroll-one-line-down() \n\
            :Meta<Key>d:delete-next-word() \n\
            :Meta<Key>D:kill-word() \n\
            :Meta<Key>h:delete-previous-word() \n\
            :Meta<Key>H:backward-kill-word() \n\
            :Meta<Key>\<:beginning-of-file() \n\
            :Meta<Key>\>:end-of-file() \n\
            :Meta<Key>]:forward-paragraph() \n\
            :Meta<Key>[:backward-paragraph() \n\
            ~Shift Meta<Key>Delete:delete-previous-word() \n\
             Shift Meta<Key>Delete:backward-kill-word(|) \n\
            ~Shift Meta<Key>Backspace:delete-previous-word() \n\
             Shift Meta<Key>Backspace:backward-kill-word(|) \n\
            <Key>Right:forward-character() \n\
            <Key>Left:backward-character() \n\
            <Key>Down:next-line() \n\
            <Key>Up:previous-line() \n\
            <Key>Delete:delete-previous-character() \n\
            <Key>BackSpace:delete-previous-character() \n\
            <Key>Linefeed:newline-and-indent() \n\
            <Key>Return:newline() \n\ <Key>:insert-char() \n\
            <FocusIn>:focus-in() \n\ <FocusOut>:focus-out() \n\
            <Btn1Down>:select-start() \n\
            <Btn1Motion>:extend-adjust() \n\
            <Btn1Up>:extend-end(PRIMARY, CUT_BUFFER0) \n\
            <Btn2Down>:insert-selection(PRIMARY, CUT_BUFFER0) \n\
            <Btn3Down>:extend-start() \n\
            <Btn3Motion>:extend-adjust() \n\
            <Btn3Up>:extend-end(PRIMARY, CUT_BUFFER0)";
```

A user-supplied resource entry can use application-specific bindings, a subset of the supplied default bindings, or both. The following is an example of a user-supplied resource entry that uses a subset of the default bindings:

```
Xmh*Text.Translations: \
    <Key>Right:forward-character() \n\
    <Key>Left:backward-character() \n\
     Meta<Key>F:forward-word() \n\
     Meta<Key>B:backward-word() \n\
```

```
:Meta<Key>]:forward-paragraph() \n\
:Meta<Key>[:backward-paragraph() \n\
<Key>:  insert-char()
```

An augmented binding that is useful with the *xclipboard* utility is:

```
*Text.Translations: #override \
    Button1 <Btn2Down>:extend-end(CLIPBOARD)
```

The Text widget fully supports the X selection and cut buffer mechanisms. The following actions can be used to specify button bindings that will cause Text to assert ownership of one or more selections, to store the selected text into a cut buffer, and to retrieve the value of a selection or cut buffer and insert it into the text value.

`insert-selection(`***name***`[,`***name***`,...])` Retrieves the value of the first (left-most) named selection that exists or the cut buffer that is not empty and inserts it into the input stream. The specified name can be that of any selection (for example, PRIMARY or SECONDARY) or a cut buffer (i.e., CUT_BUFFER0 through CUT_BUFFER7). Note that case matters.

`select-start()` Unselects any previously selected text and begins selecting new text.

`select-adjust()`

`extend-adjust()` Continues selecting text from the previous start position.

`start-extend()` Begins extending the selection from the farthest (left or right) edge.

`select-end(`***name***`[,`***name***`,...])`

`extend-end(`***name***`[,`***name***`,...])` Ends the text selection, asserts ownership of the specified selection(s), and stores the text in the specified cut buffer(s). The specified name can be that of a selection (for example, PRIMARY or SECONDARY) or a cut buffer (i.e., CUT_BUFFER0 through CUT_BUFFER7). Note that case is significant. If CUT_BUFFER0 is listed, the cut buffers are rotated before storing into buffer 0.

I.18 Toggle

The Toggle widget is an area, often rectangular, containing a text or pixmap label. This widget maintains a Boolean state (e.g. True/False or On/Off) and changes state whenever it is selected. When the pointer is on the button, the button border is highlighted to indicate that the button is ready for selection. When pointer button 1 is pressed and released, the Toggle widget indicates that it has changed state by reversing its foreground and background colors, and its `notify` action is invoked. If the pointer is moved out of the widget before the button is released, the widget reverts to its normal foreground and background colors, and releasing the button has no effect. This behavior allows the user to cancel an action.

Toggle buttons may also be part of a radio group. A radio group is a list of Toggle buttons in which no more than one Toggle may be set at any time. A radio group is identified by the widget ID of any one of its members.

The difference between a Command widget and a Toggle widget is that a Command widget typically invokes an application function when it is invoked. A Toggle widget simply changes its state (and presumably the state of some application data.) Toggles are thus useful for mailing configuration settings, which can then be applied by an associated Command widget.

I.18.1 Resources

The following new resources are associated with the Toggle widget:

`radioGroup` (class `Widget`) Specifies another Toggle widget that is in the radio group to which this Toggle widget should be added. A radio group is a group of Toggle widgets, only one of which may be set at a time. If this value is `NULL` (the default), then the Toggle is not part of any radio group and can change state without affecting any other Toggle widgets. If the widget specified in this resource is not already in a radio group, then a new radio group is created containing these two Toggle widgets. No Toggle widget can be in multiple radio groups.

`state` (class `State`) Specifies whether the Toggle widget is set (`True`) or unset (`False`). The default is `False`.

I.18.2 Translations and Actions

The following default translation bindings are used by the Toggle widget:

```
<EnterWindow>:highlight(Always)
    <LeaveWindow>:unhighlight()
    <Btn1Down>,<Btn1Up>:toggle() notify()
```

The actions supported by Toggle are listed below:

`highlight(`***`condition`***`)` Displays the internal highlight border in the color (`foreground` or `background`) that contrasts with the interior color of the Toggle widget. The conditions `WhenUnset` and `Always` are understood by this action procedure. If no argument is passed, `WhenUnset` is assumed.

`unhighlight()` Displays the internal highlight border in the color (`foreground` or `background`) that matches the interior color of the Toggle widget.

`set()` Enters the set state, in which `notify` is possible, and displays the interior of the button in the foreground color. The label or pixmap is displayed in the background color.

`unset()` Cancels the set state and displays the interior of the button, including the highlight border, in the background color. The label or pixmap is displayed in the foreground color.

`toggle()` Changes the current state of the Toggle widget, setting the widget if it was previously unset, and unsetting it if it was previously set. If the widget is to be set and is in a radio group, then this action procedure may unset another Toggle widget, causing all routines on its callback list to be invoked. The callback routines for the Toggle to be unset are called before those for the Toggle to be set.

`reset()` Cancels any `set` or `highlight` action and displays the interior of the button in the background color, with the label displayed in the foreground color.

I.18.3 Radio Groups

Two types of radio groups are typically desired by applications. In the first type, the default translations for the Toggle widget implement a "zero, or one of many" radio group. This means that no more than one button can be active, but no buttons need to be active.

The other type of radio group is "one of many" and has the more restricted policy that exactly one radio button will always be active. Toggle widgets can be used to provide this interface by modifying the translation table of each Toggle in the group:

```
<EnterWindow>: highlight(Always)
    <LeaveWindow>: unhighlight()
    <Btn1Down>,<Btn1Up>: set() notify()
```

This translation table does not allow any Toggle to be unset unless another Toggle has been set.

I.19 Viewport

The Viewport widget consists of a frame window, one or two Scrollbars, and an inner window (usually containing a child widget). The size of the frame window is determined by the viewing size of the data that is to be displayed and the dimensions to which the Viewport is created. The inner window is the full size of the data that is to be displayed and is clipped by the frame window. The Viewport widget controls the scrolling of the data directly.

When the geometry of the frame window is equal in size to the inner window, or when the data does not require scrolling, the Viewport widget automatically removes any scroll bars. The `forceBars` resource causes the Viewport widget to display any scroll bar permanently.

I.19.1 Resources

The following new resources are associated with the Viewport widget:

`allowHoriz` (class `Boolean`) Flag to allow horizontal scroll bars. Default value is `FALSE`. Setting this resource to `TRUE` allows a Viewport child to increase in size horizontally.

`allowVert` (class `Boolean`) Flag to allow vertical scroll bars. Default value is `FALSE`. Setting this resource to `TRUE` allows a Viewport child to increase in size vertically.

`forceBars` (class `Boolean`) Flag to force display of scroll bars. Default value is `FALSE`. Normally, when the geometry of the frame window is equal in size to the inner window, or when the data does not require scrolling, Viewport automatically removes any scroll bars. Setting `forceBars` to `TRUE` causes the Viewport widget to display any scroll bar permanently.

`useBottom` (class `Boolean`) Flag to indicate bottom/top bars. Default is `FALSE`, meaning to put scrollbars on top.

`useRight` (class `Boolean`) Flag to indicate right/left bars. Default is `FALSE`, meaning to put scrollbars on the left.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX J

OPEN LOOK XVIEW Toolkit Resources

This appendix describes the X resources used by XView toolkit, Sun's toolkit for producing clients that conform to the OPEN LOOK GUI. The XView toolkit is different from the Intrinsics-based toolkits (OLIT, Athena, and Motif) in that it does **not** use a widget hierarchy for assigning resources. This makes it impractical to use Resources to change the third button in the second window from skyblue to flaming-chartreuse, but we think the designers would consider that a blessing. Most of the resources are in fact concerned with the overall behavior of a client rather than with little details of one aspect of the appearance of a client.

J.1 The XView Resources

The following information is primarily obtained from Sun's documentation on the use of the XView toolkit. The section *Arguments* appears only if there are corresponding command line arguments. The sections *Type* and *Default* describe the type of argument that the option accepts. This is followed by a section describing the effect of setting the resource.

J.1.1 Window.Scale

J.1.1.1 Argument(s):

-Wx, or -scale

J.1.1.2 Type:

"small", "medium", "large", or "extra_large"

J.1.1.3 Default:

medium

Sets the initial scale of the application (larger or smaller). small is 10 pixels, medium is 12 pixels, large is 14 pixels and extra_Large is 19 pixels. The font.name resource will override the scale.

J.1.2 Font.Name

J.1.2.1 Argument(s):

-Wt, -fn, or -font

J.1.2.2 Type:

string

J.1.2.3 Default:

lucida-sans

Sets the name of the font used for the application (not control areas). To find out what fonts are available, use the xlsfonts command (see reference manual page for more information). It is also possible to see the available fonts for the Open Windows server with the *text* demo program in the demo Navigator. Start this by choosing "demos..." from the default root menu. If the font you specify cannot be found, you see will an error message such as:

```
XView warning:
Cannot load font 'galant-24' (Font package)
XView warning:
Attempting to load font 'b&h-lucida-medium-r-normal-sans-*-120-*-*-*-*-*-
            *' instead (Font package)
```

J.1.3 Window.Width and Window.Height

J.1.3.1 Argument(s):

-Ws, or -size

J.1.3.2 Type:

integer integer

J.1.3.3 Default:

depends

Sets the width and height of the application's base frame. The values are in pixels.

J.1.4 Window.X and Window.Y

J.1.4.1 Argument(s):

-Wp, or -position

J.1.4.2 Type:

integer

J.1.4.3 Default:

depends on window manager

Sets the initial position of the application's base frame in pixels. See Chapter 3, *Opening Additional Windows*, for information on specifying window geometry.

J.1.4.4 Argument(s):

-WG, or -geometry

J.1.4.5 Type:

X Geometry specification.

J.1.4.6 Default:

depends on window manager

This sets both the size and the placement of the application's base frame. This option has priority over the -size and -position arguments. See Chapter 3, *Opening Additional Windows*, for information on the format and meaning of the X Geometry string.

J.1.5 Icon.X Icon.Y

J.1.5.1 Argument(s):

-WP, -icon_position

J.1.5.2 Type:

integer integer

J.1.5.3 Default:

depends on window manager

Sets the position of the application's icon in pixels. Uses the same semantics as -position for base frames.

J.1.6 Window.Header

J.1.6.1 Argument(s):

-Wl, -label, or -title

J.1.6.2 Type:

string

J.1.6.3 Default:

depends on the application

Sets a default label for the base frame's header.However, the application can overwrite this setting and display its own header.

J.1.7 Window.Iconic

J.1.7.1 Argument(s):

-Wi, and +Wi

J.1.7.2 Type:

boolean

J.1.7.3 Default:

+Wi

These options control how an application will come up, open or closed (iconified).

J.1.8 Window.Color.Foreground

J.1.8.1 Argument(s):

-Wf, or -foreground_color

J.1.8.2 Type:

integer integer integer

J.1.8.3 Default:

0 0 0

See description in -Wb below.

J.1.9 Window.Color.Background

J.1.9.1 Argument(s):

-Wb, or -background

J.1.9.2 Type:

integer integer integer

J.1.9.3 Default:

255 255 255

These options allow the user to specify the foreground color (e.g., the color of the text in a textsw), or the background color (e.g., the color that the text is painted on) of an application. The three values should be integers between 0 and 255. They specify the amount of red, green and blue that is in the color.See -fg and -bg below for information on similar functions.

J.1.10 Window.Color.Foreground

J.1.10.1 Argument(s):

-fg, or -foreground

J.1.10.2 Type:

string (color name, or hexadecimal color specification)

J.1.10.3 Default:

black

See description in -bg below.

J.1.11 Window.Color.Background

J.1.11.1 Argument(s):

-bg, or -background

J.1.11.2 Type:

string (color name, or hexadecimal color specification)

J.1.11.3 Default:

white

These options are similar to the -Wf and -Wb options, except that they take a color argument in the form of a predefined color name (lavender, grey, goldenrod, etc.) from $OPENWINHOME/lib/rbg.txt, or a hexadecimal representation. The hexadecimal representation is of the form pound sign (#) followed by the hexadecimal representation of the red, green and blue aspects of the color.

J.1.12 Icon.Pixmap

J.1.12.1 Argument(s):

-WI, or -icon_image

J.1.12.2 Type:

string

J.1.12.3 Default:

depends on application

Sets the default filename for the icon's image.However, the application can overwrite this setting and display its own icon image. The file must be in XView icon format. The program iconedit will allow one to create an image in the icon format. Several icons have been provided in the directory $OPENWINHOME/include/images. By convention, icon format files end with the suffix ".icon".

J.1.13 Icon.Footer

J.1.13.1 Argument(s):

-WL, or -icon_label

J.1.13.2 Type:

string

J.1.13.3 Default:

depends on application

Sets a default label for the base frame's icon. However, the application can overwrite this setting and display its own icon label.

J.1.14 Icon.Font.Name

J.1.14.1 Argument(s):

-WT, or -icon_font

J.1.14.2 Type:

string

J.1.14.3 Default:

depends

Sets the name of the font used for the application's icon. To find out what fonts are available, use xlsfonts (see reference manual page for more information).

J.1.15 Window.Synchronous

J.1.15.1 Argument(s):

-sync or -synchronous, and +sync or +synchronous

J.1.15.2 Type:

boolean

J.1.15.3 Default:

+synchronous

These options allow you to make the connection that the application has with the X11 server either synchronous (-sync) or asynchronous (+sync).

J.1.16 Server.Name

J.1.16.1 Argument(s):

-Wr, or -display

J.1.16.2 Type:

string (host:display{.screen})

J.1.16.3 Default:

taken from the DISPLAY environment variable

Sets the name of the X11 server on which to connect. host is the name or address of the machine on whose server you have permission to display. display is a number corresponding to the server on which to display for that machine, and screen corresponds to which screen for the server. See reference manual page on xhost for more details on adding to permissions list.

J.1.17 Window.Mono.DisableRetained

J.1.17.1 Argument(s):

-Wdr, or -disable_retained

J.1.17.2 Type:

boolean

J.1.17.3 Default:

Not Retained on color systems, and Retained on monochrome systems

This option is useful for applications running on a monochrome display, where server memory is at a minimum. For performance reasons, monochrome windows are by default retained by the server. Using retained windows will use more memory in the X11 server; however, it also speeds up repainting when the window is covered and uncovered by other windows. When true, monochrome windows are not retained, thus saving server memory.

J.1.18 Fullscreen.Debug

J.1.18.1 Argument(s):

-Wfsdb, or -fullscreendebug

J.1.18.2 Type:

boolean

J.1.18.3 Default:

FALSE

Enables/disables fullscreen debugging mode during which XGrabs (XGrabServer(), XGrabKeyboard(), XGrabPointer()) are not done. When using FULLSCREEN, the X11 server will be grabbed which prevents other windows on the server from responding until the grab has been released by the one window which initiated the grab. Refer to the Appendix F in the XView Reference Manual: Converting SunView Applications for further details.

J.1.19 Fullscreen.Debugserver

J.1.19.1 Argument(s):

-Wfsdbs, or -fullscreendebugserver

J.1.19.2 Type:

boolean

J.1.19.3 Default:

FALSE

Enables/disables server grabbing (XGrabServer()) that is done via the fullscreen package. Refer to Appendix F in the *XView Reference Manual: Converting SunView Applications* for further details.

J.1.20 Fullscreen.Debugkbd

J.1.20.1 Argument(s):

-Wfsdbk, or -fullscreendebugkbd

J.1.20.2 Type:

boolean

J.1.20.3 Default:

FALSE

Enables/disables keyboard grabbing (XGrabKeyboard()) that is done via the fullscreen package. Refer to Appendix F in the *XView Reference Manual: Converting SunView Applications* for further details.

J.1.21 Fullscreen.Debugptr

J.1.21.1 Argument(s):

-Wfsdbp, or -fullscreendebugptr

J.1.21.2 Type:

boolean

J.1.21.3 Default:

FALSE

Enables/disables pointer grabbing (XGrabPointer()) that is done via the fullscreen package. Refer to Appendix F in the *XView Reference Manual: Converting SunView Applications* for further details.

J.1.22 Window.ReverseVideo

J.1.22.1 Argument(s)

-rv (or -reverse), and +rv (or +reverse)

J.1.22.2 Type:

boolean

J.1.22.3 Default:

False

These options control whether the foreground and background colors of the application will be reversed. If True, the foreground and background colors will be swapped. The -rv flag will set this to True, while the +rv will set it to False. This is really only useful on monochrome displays.

J.1.23 window.synchronous, +sync -sync

J.1.23.1 Values:

True, False (False)

Useful when debugging or tracking down a problem since the error codes emitted from Xlib will correspond to the immediate request made. Running in a synchronous mode will cause the application to run significantly slower.

J.1.24 mouse.modifier.button2

J.1.24.1 Values:

Shift, Ctrl, any valid modifier keysym (Shift)

When using a mouse with less than three buttons, this resource gets an equivalent mapping for the second button which is the ADJUST button on a three button mouse. For more information on keysyms, see the xmodmap reference manual page, Xlib documentation, and the include file $OPENWINHOME/include/X11/Xkeymap.h.

J.1.25 mouse.modifier.button3

J.1.25.1 Values:

Shift, Ctrl, any valid modifier keysym (Ctrl)

When using a mouse with less than three buttons, this resource gets an equivalent mapping for the third button which is the MENU button on a three button mouse. For more information on keysyms, see the xmodmap reference manual page, Xlib documentation, and the include file $OPENWINHOME/include/X11/Xkeymap.h.

J.1.26 OpenWindows.beep (Props)

J.1.26.1 Values:

never, notices, always (always)

When the value is notices, the audible bell will ring only when a notice pops up. When the value is never, the audible bell will never ring. When the value is always, the audible bell will always ring when the bell function is called by a program.

J.1.27 alarm.visible

J.1.27.1 Values:

True, False (True)

When ringing the bell in an XView program, flash the window as well to warn the user.

DefaultOpenWindows.windowColor (Props)

J.1.27.2 Values:

any valid X11 color specification (for example, #cccccc produces 80% grey)

Specify the base color for control areas for 3-D look. Takes hexadecimal representation. Three other colors used for shading and highlighting are calculated based upon the value of the specified control color. The actual calculated values are done by the OLGX library to provide a consistent color calculation between XView and OLWM. The desktop properties program allows a full range of customization and previews what the chosen 3-D look will look like. Does not apply to monochrome displays.

J.1.28 OpenWindows.workspaceColor (Props)

J.1.28.1 Values:

any valid X11 color specification (#cccccc-80% grey)

Specifies the color for the root window and the background color for icons that blend into the desktop.

J.1.29 xview.icccmcompliant

J.1.29.1 Values:

True, False (True)

When False, tell XView to set window manager hints in a way that was used before the ICCCM was adopted. Useful for window managers that are released before X11R4. Not needed with the Open Look Window Manager provided with Open Windows.

J.1.30 OpenWindows.3DLook.Color

J.1.30.1 Values:

True, False (True on all but monochrome screens)

When False, do not use the 3-D look on a color or grayscale screen.

J.1.31 OpenWindows.dragRightDistance (Props)

J.1.31.1 Values:

N (100)

Used by menus to determine when a pullright submenu would display when dragging over the menu item near a submenu. N is an integer greater than 0. A reasonable value might start at 20 and go to 200 or so. May need to try different values to see what feels right to each person.

J.1.32 Selection.Timeout

J.1.32.1 Values:

N (3)

Selection timeout value. N indicates the number of seconds that a requestor or a selection owner waits for a response.

J.1.33 OpenWindows.MouseChordMenu

J.1.33.1 Values:

True, False (False)

Turns on the mouse chording mechanism. Mouse chording was implemented to make XView work with two button mice. Holding the SELECT and the ADJUST buttons together will act as MENU button.

J.1.34 OpenWindows.MouseChordTimeout

J.1.34.1 Values:

N (100)

Mouse chording time-out value. N is in micro-seconds.

J.1.35 OpenWindows.SelectDisplaysMenu (Props)

J.1.35.1 Values:

True, False (False)

When True, the SELECT button (usually left mouse) will display the menu as well as the MENU button (usually right mouse).

J.1.36 OpenWindows.popupJumpCursor (Props)

J.1.36.1 Values:

True, False (False)

When False, do not warp the mouse to the notice when it appears.

J.1.37 notice.beepCount

J.1.37.1 Values:

N (1)

Where N is an integer to specify how many times to ring the bell when a notice appears. Ringing the bell can consist of either an audible beep and/or a visual flash.

J.1.38 OpenWindows.scrollbarPlacement (Props)

J.1.38.1 Values:

Left, Right (Right)

When set to Left, put all scrollbars on the lefthand side of the window or object.

J.1.39 OpenWindows.multiClickTimeout (Props)

J.1.39.1 Values:

N (4)

Where N is an integer greater than 2. Set the number of tenths of a second between clicks for a multi-click. A click is button-down, button-up pair.

J.1.40 text.delimiterChars

J.1.40.1 Values:

string('\011!\"#$%&\'()*+,./:;<=>?@[\\]^_`{|}~')

This resource allows the user to select the delimiter characters that are used when doing word level selections in the XView package. It was added because of the needs of the international marketplace, and it allows the user to define the local delimiters for the character set that is being used with the current keyboard and Sun workstation.

This resource is provided as a bridge until automatic selection of these characters becomes available.

Note that the octal characters can be scrambled by Xrm during a rewrite of the value of text.delimiter.Chars. Xrm interprets the text.delimiterChar string when it is loaded. Specifically it will decode the backslashed portions of the string and convert them to octal representations. When this is passed to the client application, the logic will function correctly. However, this misbehavior of Xrm causes the string to be stored incorrectly if the user saves the *.Xdefaults* file using the Xrm content of the string. The specific problem(s) that occur are the stripping of the backslash characters and the expansion of the tab character ("\t").

To correct this problem, one can put the text.delimiterChar entry into an *.Xdefaults* file that will not be overwritten when saving the workspace properties (for example, a system wide defaults file). Or a copy of the text.delimiterChar entry can be inserted after *.Xdefaults* file saves.

J.1.41 scrollbar.jumpCursor (Props)

J.1.41.1 Values:

True, False (True)

When False, the scrollbar will not move the mouse pointer when scrolling.

J.1.42 scrollbar.repeatDelay

J.1.42.1 Values:

N (100)

Where N is some integer greater than 2. Specifies the time in milliseconds when a click becomes a repeated action.

J.1.43 scrollbar.pageInterval

J.1.43.1 Values:

N (100)

Where N is some integer greater than 2. Specifies the time in milliseconds between repeats of a single page scroll.

J.1.44 scrollbar.lineInterval

J.1.44.1 Values:

N (1)

Where N is some integer greater than 0. Specifies the time in milliseconds between repeats of a single line scroll. How long to pause scrolling when holding down the SELECT button on the scrollbar elevator. Scrollbar sets up a timer routine for repeats.

J.1.45 keyboard.deleteChar

J.1.45.1 Values:

C (177 = octal for Delete)

Where C is some character either typed into an editor or specified with an octal equivalent. Specifies the delete character. This resource applies to text windows only and not to panel text items. This would work in either cmdtool or textedit or the compose window of mailtool.

J.1.46 keyboard.deleteWord

J.1.46.1 Values:

C (27 = octal for ^W)

Where C is some character either typed into an editor or specified with an octal equivalent. Specifies the delete word character. This resource applies to text windows only and not to panel text items. This would work in either cmdtool or textedit or the compose window of mailtool.

J.1.47 keyboard.deleteLine

J.1.47.1 Values:

C

Where C is some character either typed into an editor or specified with an octal equivalent. Specifies the delete line character. This resource applies to text windows only and not to panel text items. This would work in either cmdtool or textedit or the compose window of mailtool.

J.1.48 text.maxDocumentSize

J.1.48.1 Values:

N (2000)

Where N specifies the bytes used in memory before a text file is saved to a file on disk. Once this limit is exceeded, the text package will send a notice to the user to tell them that no more insertions are possible. If the file being edited is saved to a file, or it is a disk file being edited, then the limit does not apply.

J.1.49 text.retained

J.1.49.1 Values:

True, False (False)

If True, retain text windows with server backing store.

J.1.50 text.extrasMenuFilename

J.1.50.1 Values:

filename (/usr/lib/.text_extras_menu)

Where filename is an absolute location to a file. Can also be set via environment variable EXTRASMENU. This file is used for the text package's Extras menu. The commands specified in the extras menu are applied to the contents of the current selection in the textsw window and then it inserts the results at the current insertion point.

J.1.51 text.enableScrollbar

J.1.51.1 Values:

True, False (True)

When False, do not put a scrollbar on the text window.

J.1.52 text.againLimit

J.1.52.1 Values:

N (1)

Where N is an integer between 0 and 500. Number of operations the "again history" remembers for a textsw.

J.1.53 text.autoIndent

J.1.53.1 Values:

True, False (False)

When True, begin the next line at the same indentation as the previous line as typing in text.

J.1.54 text.autoScrollBy

J.1.54.1 Values:

N (1)

Where N is an integer between 0 and 100. Specifies the number of lines to scroll when type-in moves insertion point below the view.

J.1.55 text.confirmOverwrite

J.1.55.1 Values:

True, False (True)

When False, do not give user confirmation if a save will overwrite an existing file.

J.1.56 text.displayControlChars

J.1.56.1 Values:

True, False (True)

When False, use an up arrow plus a letter to display the control character instead of the character that is available for the current font.

J.1.57 text.undoLimit

J.1.57.1 Values:

N (50 maximum of 500)

Where N is an integer between 0 and 500. How many operations to save in the undo history log. These operations will be undone when you press the "Undo" key in the text window.

J.1.58 text.insertMakesCaretVisible

J.1.58.1 Values:

If_auto_scroll (Always)

Controls whether insertion causes repositioning to make inserted text visible.

J.1.59 text.lineBreak

J.1.59.1 Values:

Clip, Wrap_char, Wrap_word (Wrap_word)

Determines how the textsw treats file lines when they are too big to fit on one display line.

J.1.60 text.margin.bottom

J.1.60.1 Values:

N (0)

Where N is an integer between -1 and 50. Specifies the minimum number of lines to maintain between insertion point and bottom of view. A value of -1 turns auto scrolling off.

J.1.61 mouse.multiclick.space

J.1.61.1 Values:

N (4)

Where N is an integer between 2 and 500. Specifies the maximum number of pixels between successive mouse clicks to still have the clicks considered as a multi-click event.

J.1.62 text.storeChangesFile

J.1.62.1 Values:

True, False (True)

When False, do not change the name of the current file being edited to the name of the file that is stored. The name of the current file is reflected in the titlebar of the textedit frame.

J.1.63 text.margin.top

J.1.63.1 Values:

N (2)

Where N is an integer between -1 and 50. Specifies the minimum number of lines to maintain between the start of the selection and the top of the view. A value of -1 means defeat normal actions.

J.1.64 text.margin.left

J.1.64.1 Values:

N (8)

Where N is an integer between 0 and 2000. Specifies the margin in pixels that the text should maintain between the left hand border of the window and the first character on each line.

J.1.65 text.margin.right

J.1.65.1 Values:

N (0)

Where N is an integer between 0 and 2000. Specifies the margin in pixels that the text should maintain between the right hand border of the window and the last character on each line.

J.1.66 text.tabWidth

J.1.66.1 Values:

N (8)

Where N is an integer between 0 and 50. Specifies the width in characters of the tab character.

J.1.67 term.boldStyle

J.1.67.1 Values:

None, Offset_X, Offset_Y, Offset_X_and_Y, Offset_XY, Offset_X_and_XY, Offset_Y_and_XY, Offset_X_and_Y_and_XY, Invert (Invert)

Specify the text emboldening style for a terminal based window.

J.1.68 term.inverseStyle

J.1.68.1 Values:

Enable, Disable, Same_as_bold (Enable)

Specify the text inverting style for a terminal based window.

J.1.69 term.underlineStyle

J.1.69.1 Values:

Enable, Disable, Same_as_bold (Enable)

Specify the text underlining style for a terminal based window.

J.1.70 term.useAlternateTtyswrc

J.1.70.1 Values:

True, False (True)

When True, and a $HOME/.ttyswrc is not found, look for an alternate ttyswrc file. When False, do not look for an alternate file is one is not found in the home directory, $HOME/.ttyswrc.

J.1.71 term.alternateTtyswrc

J.1.71.1 Values:

filename ($XVIEWHOME/lib/.ttyswrc)

Where filename specifies a complete filename and absolute path of an alternate ttyswrc file. This is only used if a .ttyswrc file is not found in $HOME/.ttyswrc and term.useAlternateTtyswrc is True.

J.1.72 term.enableEdit

J.1.72.1 Values:

True, False (True)

When False, do not keep an edit log of what has been typed into the term window. This is set to false automatically when switching from a scrollable term to one that is not scrollable.

J.2 Internationalized Command Line Resource Arguments

The following command line arguments are relevant to internationalization. Locale refers to the language and cultural conventions used in a program. Locale setting is the method by which the language and cultural environment of a system is set. Locale setting affects the display and manipulation of language-dependent features.

J.2.1 basicLocale

J.2.1.1 Argument(s):

-lc_basiclocale

J.2.1.2 Type:

string

J.2.1.3 Default:

"C"

Specifies the basic locale category, which sets the country of the user interface.

J.2.2 displaylang

J.2.2.1 Argument(s):

-lc_displaylang

J.2.2.2 Type:

string

J.2.2.3 Default:

“C”

Specifies the displaylanguagelocale category, sets the language in which labels, messages, menu items, and help text are displayed.

J.2.3 inputLang

J.2.3.1 Argument(s):

-lc_inputlang

J.2.3.2 Type:

string

J.2.3.3 Default:

"C"

Specifies the inputlanguagelocale category, sets the language used for keyboard input.

J.2.4 numeric

J.2.4.1 Argument(s):

-lc_numeric

J.2.4.2 Type:

string

J.2.4.3 Default:

"C"

Specifies the numeric locale category, which defines the language used to format numeric quantities.

J.2.5 timeFormat

J.2.5.1 Argument(s):

-lc_timeformat

J.2.5.2 Type:

string

J.2.5.3 Default:

"C"

Specifies the time format locale category, which defines the language used to format time and date.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX K

OPEN LOOK Mouseless Keyboard Summary

The OPEN LOOK specification suggests that implementations should provide a mode for mouseless operations, for those most disposed to use a keyboard for everything. While one wonders what such people will do when computers need to speak and provide video, there is ample precedent for such "mouseless operations".—it wasn't that long ago that the average MIS Manager didn't even know that some computers *had* mice. Well, anyway, the OPEN LOOK GUI provides for mouseless work by defining a series of tags and assigning key values to them. For example, the tag ACTION_CUT is used to Cut the selected objects. On OpenWindows, it is initially mapped to x+Meta,L10", which means that either pressing the L10 key, or holding the "Meta" key while pressing "x", will cut the current selection. And, you can change this by setting the X Resource named "Cut" to some other value.

K.1 AT&T Mouseless Operations

The AT&T mouseless operations are not documented with their resources. Instead, the AT&T-OL version of the *properties* editor allows you to change all the common mouseless operation key combinations, and this has already been documented. For this reason, the AT&T-OL mouseless keys are not listed here; please refer to the AT&T documentation for this information.

K.2 OpenWindows Mouseless Operations

The following table lists the Sun mouseless mode operations; most of the names of the actions are self-explanatory.

Example K-1. OpenWindows Mouseless Keys

OPEN LOOK Name	OpenWindows Resource Name (`OpenWindows.KeyboardCommand.`)	Default Binding
ACTION_ADJUST	Adjust	Insert+Alt
ACTION_AGAIN	Again	a+Meta,a+Ctrl+Meta,L2
ACTION_COPY	Copy	c+Meta,L6
ACTION_COPY_THEN_PASTE	CopyThenPaste	p+Meta
ACTION_CUT	Cut	x+Meta,L10
ACTION_DATA_END	DataEnd	End+Ctrl,R13+Ctrl
ACTION_DATA_END	DataEnd	End,R13,Return+Ctrl,End+Shift
ACTION_DATA_START	DataStart	Home+Ctrl,R7+Ctrl
ACTION_DATA_START	DataStart	Home,R7,Return+Shift+Ctrl,Home+Shift
ACTION_DEFAULT_ACTION	DefaultAction	Return+Meta
ACTION_DOWN	Down	Down
ACTION_DOWN	Down	n+Ctrl,P+Ctrl,Down,R14,Down+Shift
ACTION_EMPTY	Empty	e+Meta,e+Ctrl+Meta
ACTION_ERASE_CHAR_BACKWARD	EraseCharBackward	Delete,BackSpace
ACTION_ERASE_CHAR_FORWARD	EraseCharForward	Delete+Shift,BackSpace+Shift
ACTION_ERASE_CHAR_FORWARD	EraseCharForward	Delete+Shift,BackSpace+Shift
ACTION_ERASE_LINE	EraseLine	Delete+Meta,BackSpace+Meta

Example K-1. OpenWindows Mouseless Keys

OPEN LOOK Name	OpenWindows Resource Name (`OpenWindows.KeyboardCommand.`)	Default Binding
ACTION_ERASE_LINE_BACKWARD	EraseLineBackward	u+Ctrl
ACTION_ERASE_LINE_END	EraseLineEnd	U+Ctrl
ACTION_ERASE_WORD_BACKWARD	EraseWordBackward	w+Ctrl
ACTION_ERASE_WORD_FORWARD	EraseWordForward	W+Ctrl
ACTION_FIND_BACKWARD	FindBackward	F+Meta,L9+Shift
ACTION_FIND_FORWARD	FindForward	f+Meta,L9
ACTION_GO_LINE_FORWARD	GoLineForward	apostrophe+Ctrl,R11
ACTION_GO_PAGE_BACKWARD	GoPageBackward	R9
ACTION_GO_PAGE_FORWARD	GoPageForward	R15
ACTION_GO_WORD_FORWARD	GoWordForward	slash+Ctrl,less+Ctrl
ACTION_HELP	Help	Help
ACTION_HORIZONTAL_SCROLLBAR_MENU	HorizontalScrollbar-Menu	h+Alt
ACTION_INCLUDE_FILE	IncludeFile	i+Meta
ACTION_INPUT_FOCUS_HELP	InputFocusHelp	question+Ctrl
ACTION_INSERT	Insert	Insert
ACTION_JUMP_DOWN	JumpDown	Down+Ctrl
ACTION_JUMP_LEFT	JumpLeft	Left+Ctrl
ACTION_JUMP_LEFT	JumpLeft	comma+Ctrl,greater+Ctrl
ACTION_JUMP_MOUSE_TO_INPUT_FOCUS	JumpMouseToInput-Focus	j+Alt
ACTION_JUMP_RIGHT	JumpRight	Right+Ctrl
ACTION_JUMP_RIGHT	JumpRight	period+Ctrl

Example K-1. OpenWindows Mouseless Keys

OPEN LOOK Name	OpenWindows Resource Name (`OpenWindows.KeyboardCommand.`)	Default Binding
ACTION_JUMP_UP	JumpUp	Up+Ctrl
ACTION_LEFT	Left	Left
ACTION_LEFT	Left	b+Ctrl,F+Ctrl,Left,R10,Left+Shift
ACTION_LINE_END	LineEnd	e+Ctrl,A+Ctrl
ACTION_LINE_START	LineStart	a+Ctrl,E+Ctrl
ACTION_LOAD	Load	l+Meta
ACTION_MATCH_DELIMITER	MatchDelimiter	d+Meta
ACTION_MENU	Menu	space+Alt
ACTION_MORE_HELP	MoreHelp	Help+Shift
ACTION_MORE_TEXT_HELP	MoreTextHelp	Help+Shift+Ctrl
ACTION_NEXT_ELEMENT	NextElement	Tab+Ctrl
ACTION_NEXT_PANE	NextPane	a+Alt
ACTION_PANEL_END	PanelEnd	bracketright+Ctrl
ACTION_PANEL_START	PanelStart	bracketleft+Ctrl
ACTION_PANE_BACKGROUND	PaneBackground	b+Alt
ACTION_PANE_DOWN	PaneDown	R15
ACTION_PANE_LEFT	PaneLeft	R9+Ctrl
ACTION_PANE_RIGHT	PaneRight	R15+Ctrl
ACTION_PANE_UP	PaneUp	R9
ACTION_PASTE	Paste	v+Meta,L8
ACTION_PREVIOUS_ELEMENT	PreviousElement	Tab+Shift+Ctrl
ACTION_PREVIOUS_PANE	PreviousPane	A+Alt
ACTION_PROPS	Props	L3
ACTION_QUOTE_NEXT_KEY	QuoteNextKey	q+Alt

Example K-1. OpenWindows Mouseless Keys

OPEN LOOK Name	OpenWindows Resource Name (`OpenWindows.KeyboardCommand.`)	Default Binding
ACTION_RESUME_MOUSELESS	ResumeMouseless	Z+Alt
ACTION_RIGHT	Right	Right
ACTION_RIGHT	Right	f+Ctrl,B+Ctrl,Right,R12,Right +Shift
ACTION_ROW_END	RowEnd	End,R13
ACTION_ROW_START	RowStart	Home,R7
ACTION_SCROLL_DATA_END	ScrollDataEnd	End+Alt+Ctrl,R13+Alt+Ctrl
ACTION_SCROLL_DATA_START	ScrollDataStart	Home+Alt+Ctrl,R7+Alt+Ctrl
ACTION_SCROLL_DOWN	ScrollDown	Down+Alt
ACTION_SCROLL_JUMP_DOWN	ScrollJumpDown	Down+Alt+Ctrl
ACTION_SCROLL_JUMP_LEFT	ScrollJumpLeft	Left+Alt+Ctrl
ACTION_SCROLL_JUMP_RIGHT	ScrollJumpRight	Right+Alt+Ctrl
ACTION_SCROLL_JUMP_UP	ScrollJumpUp	Up+Alt+Ctrl
ACTION_SCROLL_LEFT	ScrollLeft	Left+Alt
ACTION_SCROLL_PANE_DOWN	ScrollPaneDown	R15+Alt
ACTION_SCROLL_PANE_LEFT	ScrollPaneLeft	R9+Alt+Ctrl
ACTION_SCROLL_PANE_RIGHT	ScrollPaneRight	R15+Alt+Ctrl
ACTION_SCROLL_PANE_UP	ScrollPaneUp	R9+Alt
ACTION_SCROLL_RIGHT	ScrollRight	Right+Alt
ACTION_SCROLL_ROW_END	ScrollRowEnd	End+Alt,R13+Alt
ACTION_SCROLL_ROW_START	ScrollRowStart	Home+Alt,R7+Alt
ACTION_SCROLL_UP	ScrollUp	Up+Alt
ACTION_SELECT_ALL	SelectAll	End+Shift+Meta,R13+Shift+Meta

Example K-1. OpenWindows Mouseless Keys

OPEN LOOK Name	OpenWindows Resource Name (`OpenWindows.KeyboardCommand.`)	Default Binding
ACTION_SELECT_DATA_END	SelectDataEnd	End+Shift+Ctrl,R13+Shift+Ctrl
ACTION_SELECT_DATA_START	SelectDataStart	Home+Shift+Ctrl,R7+Shift+Ctrl
ACTION_SELECT_DOWN	SelectDown	Down+Shift
ACTION_SELECT_FIELD_BACKWARD	SelectFieldBackward	Tab+Shift+Ctrl
ACTION_SELECT_FIELD_FORWARD	SelectFieldForward	Tab+Ctrl
ACTION_SELECT_JUMP_DOWN	SelectJumpDown	Down+Shift+Ctrl
ACTION_SELECT_JUMP_LEFT	SelectJumpLeft	Left+Shift+Ctrl
ACTION_SELECT_JUMP_RIGHT	SelectJumpRight	Right+Shift+Ctrl
ACTION_SELECT_JUMP_UP	SelectJumpUp	Up+Shift+Ctrl
ACTION_SELECT_LEFT	SelectLeft	Left+Shift
ACTION_SELECT_NEXT_FIELD	SelectNextField	Tab+Meta
ACTION_SELECT_PANE_DOWN	SelectPaneDown	R15+Shift
ACTION_SELECT_PANE_LEFT	SelectPaneLeft	R9+Shift+Ctrl
ACTION_SELECT_PANE_RIGHT	SelectPaneRight	R15+Shift+Ctrl
ACTION_SELECT_PANE_UP	SelectPaneUp	R9+Shift
ACTION_SELECT_PREVIOUS_FIELD	SelectPreviousField	Tab+Shift+Meta
ACTION_SELECT_RIGHT	SelectRight	Right+Shift
ACTION_SELECT_ROW_END	SelectRowEnd	End+Shift,R13+Shift
ACTION_SELECT_ROW_START	SelectRowStart	Home+Shift,R7+Shift
ACTION_SELECT_UP	SelectUp	Up+Shift
ACTION_STOP	Stop	L1

Example K-1. OpenWindows Mouseless Keys

OPEN LOOK Name	OpenWindows Resource Name (`OpenWindows.KeyboardCommand.`)	Default Binding
ACTION_STORE	Store	s+Meta
ACTION_SUSPEND_MOUSELESS	SuspendMouseless	z+Alt
ACTION_TEXT_HELP	TextHelp	Help+Ctrl
ACTION_TRANSLATE	Translate	R2
ACTION_UNDO	Undo	u+Meta,L4
ACTION_UP	Up	Up
ACTION_UP	Up	p+Ctrl,N+Ctrl,Up,R8,Up+Shift
ACTION_VERTICAL_SCROLLBAR_MENU	VerticalScrollbarMenu	v+Alt

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX L

SunView Applications under OpenWindows

If you hang around Sun users for long enough, you'll eventually hear them talking about a window system called "SunView". Before the emergence of X11 and NeWS, most users of Sun Workstations used Sun's own windowing system, called SunView. Indeed, many old-time Sun aficionados will mourn the passing of SunView. It was small and fast[1], which meant you could do useful work on a Sun-3/50 with a mere(!) four megabytes of main storage. And on my SparcStation 1 (the classic SPARC machine) running SunView, *cmdtool* starts up in about one second, while an XView *cmdtool* or an *xterm* takes slightly over two seconds[2]. Most of this overhead is probably due to the networked nature of communication between the program, the X server, and the X window manager. SunView was a full-featured system, with its own (the original) version of *shelltool/cmdtool*, desk accessories such as *perfmeter*, etc. There was even (in SunOS 4.1.1) a SunView *fileview* program, which is similar to the File Manager discussed in Chapter 4, *Using the OPEN LOOK File Manager*. But despite its speed, SunView wasn't able to muster support in the days of networked window systems, so it has had to pass on to that great swap device in the sky.

But wait; SunView lives on! To make OpenWindows serve as a complete working environment for users of modern graphical programs, Sun Microsystems provides full compatibility with applications written for their first-generation window system, SunView. OpenWindows 2.0 introduced, and 3.x continues, the tradition of providing reasonable backwards compatibility for SunView applications. While few new applications are being developed for SunView, if you are making extensive use of a variety of programs on your Sun, you may meet up with a SunView application. Thus you should at least know the basics of running SunView applications under OpenWindows. This appendix shows you

1. But still not as fast as MGR, the free software window system from Steve Uhler of Bellcore. It's fast, but there is no real body of software accumulating for it as there is for The X Window System.

2. All cases were timed after one of these windows was started, to rule out disk- and cache-induced delays

how to use SunView applications under Sun's version of OpenWindows: what works, what doesn't, and what problems can crop up.

L.1 The Similarities

Figure L-1 shows a fairly typical native SunView window; this is how workstation screens looked before The X Window System came along. Like an X display, the screen has two *shelltool* windows, a small one for console messages and a larger one for working in, and a *perfmeter* and a clock. As well, this screendump shows the root menu pulled down to show the items for starting clock windows.

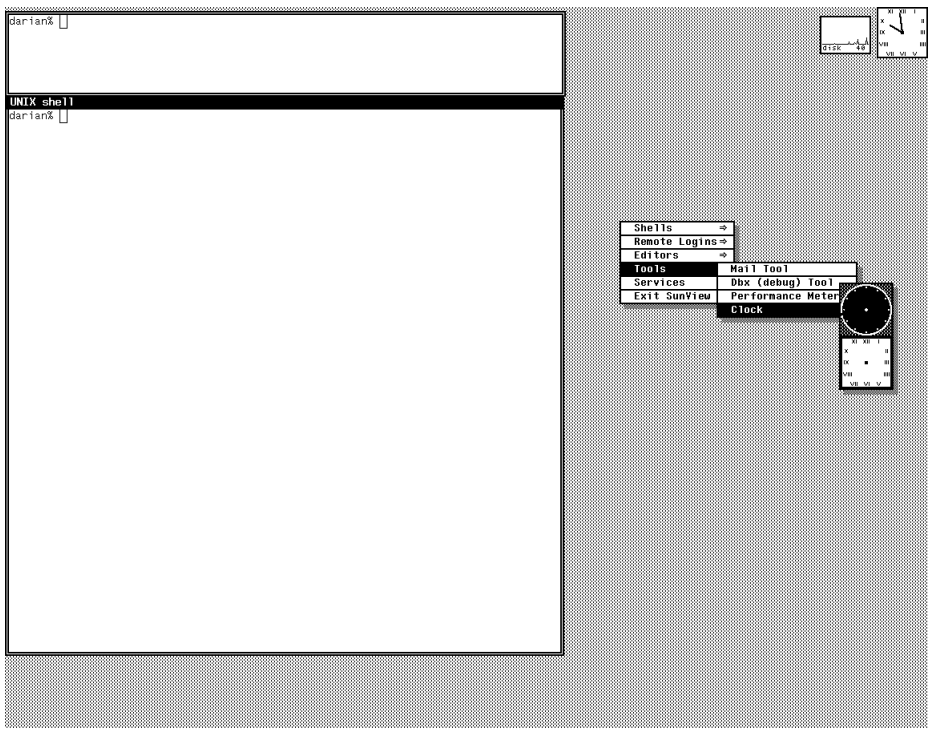


Figure L-1. Typical SunView **Display.**

Notice the following similarities between SunView and OpenWindows:

- there can be arbitrarily many windows, and they overlap in about the same way
- most of the time you do most of your work in one or more shell windows
- *cmdtool/shelltool* is almost identical to its OpenWindows descendants. Even the escape codes in the SunView terminal windows are the same as those for *cmdtool*/*shelltool* in Appendix E, *Control Sequences for xterm and cmdtool*.

- Cut-and paste is similar, but the terminal pane menu has the added item *send* (combines copy and paste) menu item, which to this day is distressingly absent from the OpenWindows versions (the functionality is there, but not in the menus; see Chapter 2, *Working in the OPEN LOOK Environment*).
- tools like clocks and *perfmeter*s are almost identical to their OpenWindows descendants
- menus are activated by depressing the right button
- root menu, submenus. Menu contents controlled by files, similar in syntax to OPEN LOOK Window Manager
- There are special editors (SunView's *defaultsedit* is the ancestor of OpenWindows's *properties* editor).

L.2 The Differences

The SunView window system was organized quite differently than is X11. It is not a networked window system; it is tightly interwoven with the UNIX kernel, and there is no separate "window manager" client at all. This means that you cannot change the overall behavior nor the "look and feel" of SunView clients, as you can in X11 by changing window managers. And you use "defaults files", not X Resources, to customize SunView programs.

L.2.1 Always at the forefront...

Pity the poor OpenWindows server program! It has to keep track not only of X11 programs' windows, and NeWS programs' windows, but also SunView program windows. X11 and NeWS are similar enough that they can co-exist on one screen. But SunView programs do not respond to the same kind of events, and do not have the same overall behavior, so they cannot really live on the same screen. To sort this out, the OpenWindows server maintains the fiction of an "overlay plane", or separate screen level, to run SunView applications in. The result is that SunView applications will always appear to be "in front of" all X11 and NeWS applications. To visually remind you that a given program is running in SunView mode, the SunView program's windows are surrounded by a fat white border. Figure L-2 is an example of a SunView version of the *dbxtool* program debugger: running in the OpenWindows environment. Notice how the border around the dbxtool window "cuts into" the windows around it, including the Workspace menu, the *xv* control panel being used to record the screen, and the borders of the terminal windows, with a large white space around it. This is your reminder that a SunView application is in use, and that certain things (notably pointer and menu operations) must be done differently. These are described in the next section.

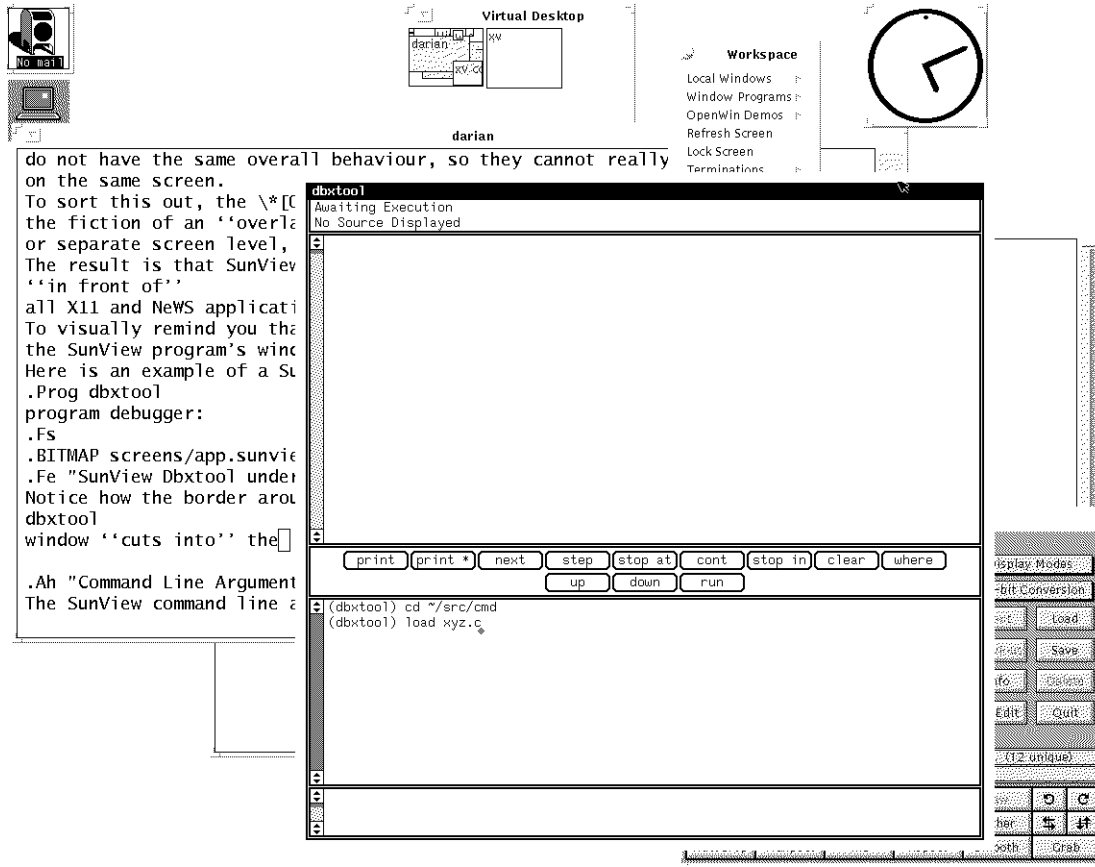


Figure L-2. SunView dbxtool in front of OpenWindows windows.

L.3 Pointer and Menu Conventions

Many of the pointer and menu button conventions are similar or identical to those of OpenWindows. Indeed, when the pointer is not in a SunView window, it behaves "normally". When in a SunView window, titlebar, or icon, however, there are a few differences.

L.3.1 Pointer button conventions

Table L-1 is a summary of the pointer button conventions used when running SunView applications under OpenWindows.

Table L-1. SunView Button Functions

Where	Button	Function
Root window	All Buttons	OPEN LOOK conventions (controlled by window manager)

Table L-1. SunView Button Functions

Where	Button	Function
In Titlebar	Button 1	Front
	Button 2	Move
	Button 3	Window menu
In Application Window	Button 1	Program-defined
	Button 2	Program-defined
	Button 3	Usually client menu

L.3.2 The Root Menu

When the pointer is on the OpenWindows background or root window, its behavior is entirely controlled by the window manager. Running the recommended *olwm* or *olvwm* will cause the normal OPEN LOOK conventions to apply to all keyboard and pointer button activity in the root window.

L.3.3 The Window Menu

Under X11, the "window manager" program provides the "window menu", the menu seen when clicking the MENU button on the titlebar of a window. Under SunView, the user program provides the window menu, so you may see variations from one program to another. The typical program provides the menu shown in Figure L-3, when not iconified.

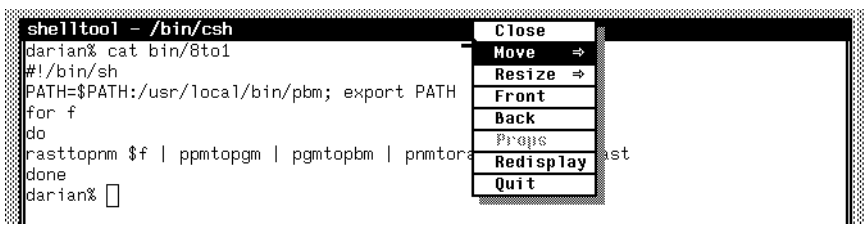


Figure L-3. Typical SunView **Window Menu.**

Most of these entries have the same meaning as they do under OPEN LOOK.

One difference is in what happens when you click on an icon - click-to-select (single-click) actually opens the icon, rather than just selecting it as most X window managers do.

L.3.4 Keyboard Shortcuts

Most of the common keyboard shortcuts are the same, including the common **L5, L7, L6, L8, L10, L4,** and **HELP.** See Appendix K, *OPEN LOOK Mouseless Operations* for list of the standard Sun keyboard shortcuts.

L.4 SunView Customization Files

As with OpenWindows, there are numerous files used to control the way programs operate. For each of these files, there is both a system-wide set of files (in */usr/lib*) and a per-user set in a user's home directory. A SunView user might have as many as five files to customize her environment, as shown in Table L-2. Some of these are used when SunView programs run under OpenWindows, and some are not.

Table L-2. SunView Files

Name	Function	Used under OpenWindows?	OpenWindows Analog
~/.defaults	Set defaults for various programs	Yes	*.Xdefaults*
~/.rootmenu	Workspace Menu	No	*~/.openwin-menu*
~/.suntools	Programs to run	No	*~/.xinitrc* or *~/.openwin-init*
~/.text_extras_menu	Menu for *Extras* in text windows	Yes	*~/.text_extras_menu*
~/.textswrc	Customize all "textsw" windows	Yes	*~/.textswrc*

L.4.1 The Defaults files vs. X11 Defaults

The SunView system used "defaults" files to let the user customize the behavior of individual window programs. Their syntax is simpler than that of X Resources—there is no "Widget Hierarchy", for example—but quite different Here is a section of one such file:

```
//Set/askbcc "No"
 $Enumeration ""
 $Help "Enables/disables prompting user for 'bcc' field when sending."
 No "No"
 No/$Help "Do NOT automatically prompt user for 'bcc' field when
           sending."
 Yes "Yes"
 Yes/$Help "Automatically prompt user for 'bcc' field when sending."

//Set/bell "0"
 $Help "Number of times to ring the audible bell when new mail arrives."
```

Part of the structure here is to dynamically control the *defaultsedit* program. This program, shown in Figure L-4,: was used to customize the behavior of SunView programs for an

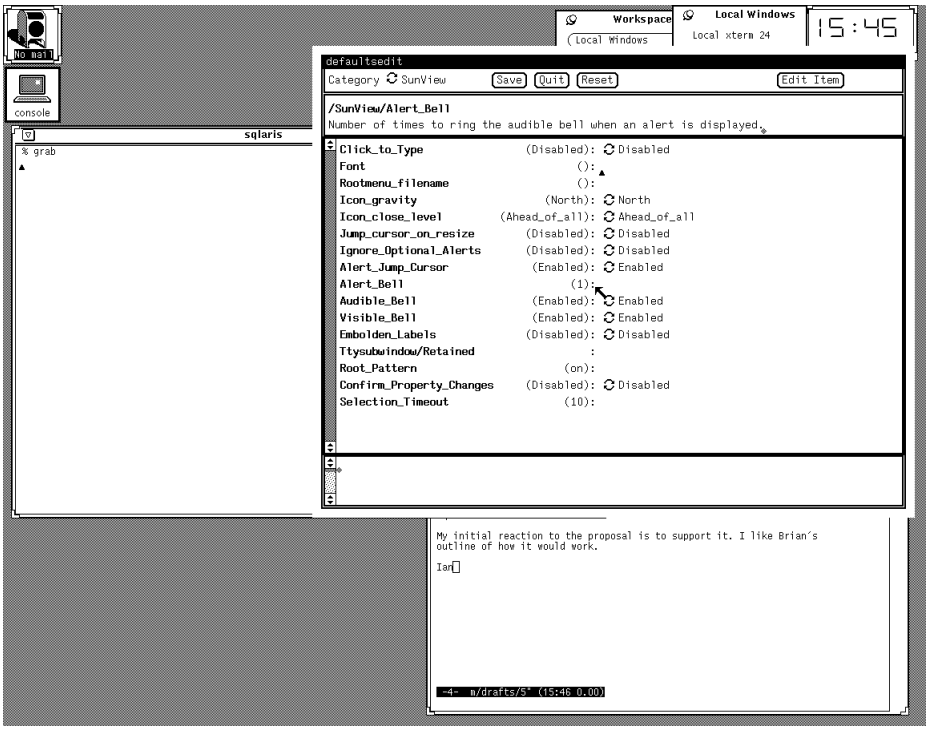


Figure L-4. Defaultsedit (SunView) in action

individual user, like the *properties* program described in Chapter 13, *Customizing olwm*. It only saved the changed values, so a user's *~/.defaults* file might look like this

```
SunDefaults_Version 2
/Mail/Set/editmessagewindow     "Yes"
/Mail/Set/hold  "Yes"
```

For users who may have existing Defaults files, there are two programs. The first, *input_from_defaults*, is normally run automatically, to read your Defaults files and apply them to the running copy of OpenWindows (see the next section for an example). The other, *convert_to_Xdefaults*, is invoked manually, and only once, when you are moving from SunView to X11 or OpenWindows. Its function is to convert a Defaults file to the corresponding X Resources format. This program is normally run with its standard output redirected into a file that will become (or be appended to) your *.Xdefaults* file. See the Reference Page in Part Three if you need to use this program.

L.5 SunView Controls

The use of buttons, menus, and other controls in SunView is very similar to that of OPEN LOOK. This is not surprising, since historically SunView is one of the most direct ancestors of the OPEN LOOK GUI. However there are a few differences that should be noted:

1. The SunView scrollbar is much more like the Athena (*xterm*) scrollbar (see Appendix A, *The xterm/olterm Terminal Emulator*) than like the OPEN LOOK scrollbar described in Chapter 2, *Working in the OPEN LOOK Environment*.
2. Instead of "abbreviated menu buttons", SunView programs uses "cyclical buttons". For example, in the *defaultsedit* window in Figure L-4, clicking the left button on the little circle beside the word *Category* will change the main category from SunView Controls through several other major categories.

By and large, however, a user of the OPEN LOOK GUI will have little trouble getting used to SunView programs.

L.6 Command Line Arguments

The SunView command line arguments are a subset of those discussed in Chapter 11, *Command Line Options*, in particular, in Table 11-2. For example, here is the list of options from the SunView version of *cmdtool*:

Table L-3. SunView Command Arguments

Flag	Long Flag	Argument Type	Notes
-Ww	-width	text columns	
-Wh	-height	text lines	
-Ws	-size	x y (pixels)	
-Wp	-position	x y	
-WP	-icon_position	x y	
-Wl	-label	(quoted) string	
-Wi	-iconic		
-Wn	-no_name_stripe		
-Wt	-font	filename	
-Wf	-foreground_color	red green blue	0-255 (no color-full color)
-Wb	-background_color	red green blue	0-255 (no color-full color)
-Wg	-set_default_color	same	(apply color to subwindows too)
-WI	-icon_image	filename	
-WL	-icon_label	(quoted) string	
-WT	-icon_font	filename	

Table L-3. SunView Command Arguments

Flag	Long Flag	Argument Type	Notes
-WH	-help		

As you can see, these options are almost all the same as the like-named options discussed in Chapter 12, but many of the ones discussed there are not accepted in SunView applications. In particular there is no *-display* option: SunView windows will only appear on the screen of the computer they are running on. And there is no "color name database" (see Section 11.5, "Specifying Color" on Page 289) so you have to specify colors by their numeric RGB values.

L.7 Setup

In order to use the SunView compatibility mode, there are some programs that you must run. If you use the provided OpenWindows-specific startup scripts, this is done automatically. If not, you should copy a few lines from the various system startup files. The provided *openwin-sys* file contains these lines:

```
# Load SunView defaults and invoke SunView/XView selection service
# if NOSunView != 1
if [ -z "$NOSunView" -o "$NOSunView" -ne 1 ];
 then input_from_defaults
 sv_xv_sel_svc &
fi
```

And the provided *.xinitrc* file *(/usr/openwin/lib/Xinitrc*) includes this:

```
# SunView binary compatibility is default mode.
 [ -z "$NOSunView" -o "$NOSunView" -ne 1 ] && eval 'svenv -env'
```

These both test the shell variable NOSunView and, if it is not present, fire up various programs that are part of the SunView compatibility mode. If you maintain your own *.xinitrcc* file, you should add these entries to it. Leaving out the testing, all you really need is:

```
input_from_defaults# get sunvew defaults
sv_xv_sel_svc &# enable cut-n-paste btwn SunView and X11
eval 'svenv -env'# setup "environment variables" for SunView compat.
```

Alternately, you can run obtain the first two commands by running the provided "system" file from within your *.xinitrc* file, just by including this line in your *.xinitrc* file.

```
$OPENWINHOME/lib/openwin-sys # OpenWin "system" initialization
```

L.8 "OPEN LOOK/SunView"

To further confuse you, a group in Sun has created an interim toolkit called "OPEN LOOK SunView" (or vice versa). This was to allow developers to build SunView applications that conform to the OPEN LOOK specification. These run as SunView applications under OpenWindows, but use more-or-less the OPEN LOOK conventions, not those shown in the tables

above. As far as can be determined, this toolkit has not been distributed outside Sun. However there are two products that are known to use it, and there may be others:

1. The "DeskSet Environment for SunView", part number DSK-1.0-4-34R-5, which gives SunView users OPEN LOOK versions of File Manager, cm, Mailtool, calctool, textedit, printtool, tapetool, snapshot, iconedit, perfmeter, binder, cmdtool, shelltool, console and Sun's clock.
2. Sun's SunNet License Platform's *nladmin* (network license administration) tool.

Unless you use either of these packages, you are unlikely to run into the "SunView OPEN LOOK".

L.9 Future Directions

The SunView compatibility mode provides a useful transition tool to allow users of OpenWindows to run any of the hundreds of SunView applications that they may have available.This compatibility works in OpenWindows Release 3.1 under SunOS 5.1, but will end in an unspecified future release of SunOS[1]; applications that are created under SunOS4.x *must* be dynamically linked if they are to run on SunOS 5.x/Solaris 2. You should convert your applications to X11 as soon as possible.

If you have the source code to an application available, there are (at least) two approaches. One method of converting SunView applications to X11 is to use Sun's XView toolkit; XView is intentionally similar to the SunView toolkit, to facilitate just such conversion. Simple SunView applications can usually be converted using a program provided with OpenWindows (called *convert_to_xview*, and described in the manual page in Part Three of this guide). Sophisticated applications will require some programmer time to convert. XView is described more fully in Volume Seven, *XView Programming Manual*.

Another approach is to use Sun's *devGuide* (Developer's Guide) to re-write the user interface code; GUIDE can generate the user-interface code for XView, for OLIT (an X Intrinsics-based Toolkit that is conceptually closer to Motif), or for The NeWS Toolkit. Either of these approaches – *convert_to_xview* or *devGuide* – will allow your programming staff to update older applications to work with X11 and/or OpenWindows, so that programs you have been using under SunView can continue to be used on The X Window System.

1. One can imagine that some future release of OpenWindows will replace the SunView application emulation with a mode that runs MicroSoft Windows applications in the same kind of "overlay plane" without having to start up SunPC and then MS-Windows.

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX M

Working with Motif

Since neither OPEN LOOK nor Motif has a complete corner on the X market, there will always be some applications that you want to run that aren't available with the interface you want. Since Motif is dominant, though, a gradual transition to Motif seems probable. This appendix deals with some specifics of using applications written using the Motif toolkit in an OPEN LOOK environment, and the window manager and file considerations.

M.1 Working with Motif Applications

The Athena widget set provides X Toolkit applications with certain common features, many of which were described in Chapter 8. As explained in Chapter 1, an application coded using the Motif widget set has a different look and feel.

In this Appendix, we'll look at some of the features you're liable to encounter in a Motif application and learn how to use them. Most of these features are provided in a slightly different flavor by the OPEN LOOK GUI.

Most of the sample components we're using are taken from the *mre* demo program described in The Motif resource editor assists you in editing your own resource specification file, but it is primarily intended to demonstrate many of the Motif widgets.

The following sections mention the comparable Athena widgets where appropriate. Some of the Athena widgets are illustrated using the standard MIT clients in Chapter 8, *Other Standard Clients*, as well as in Appendix A, *The xterm/olterm Terminal Emulator*, and Chapter 9, *Graphics Clients*.

M.1.1 Dialog Boxes and Push Buttons

If you ask an X client for any drastic action, it is supposed to prompt you for confirmation. For example, the OPEN LOOK Window Manager will prompt you with a Notice window when you try to exit. Motif applications behave similarly. For example, if you were run-

ning the Motif window manager *mwm*, and try to "restart" it from the window manager's *Root* menu, you would see the dialog box pictured in Figure M-1.

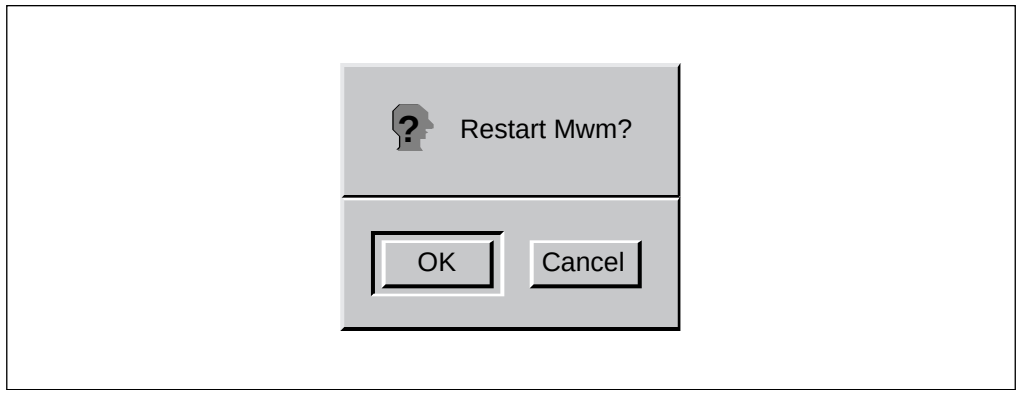


Figure M-1. Typical Motif dialog box with two push buttons

A dialog box generally displays a message relevant to the application and requires a response from the user. In this case, the dialog box queries whether you really want to `Restart mwm?`.

A Motif dialog box contains one or more *push buttons* that allow you to respond to the message. (Many applications use push buttons; they are not confined to dialog boxes.) When a dialog is displayed and the default click-to-type focus is in effect, the input focus is usually switched to the dialog window. Until you respond to the dialog box, the application cannot continue. Once you respond to the dialog, the focus should switch back to the main application window.

Whether the dialog box contains one push button or multiple buttons, one button is always highlighted, generally by outlining. You can push the highlighted push button simply by pressing the Return key on your keyboard. To push another button, you must place the pointer on it and click the first pointer button.

A response might be a simple acknowledgment that you've seen the message: some dialogs feature only one button that reads **OK**. For instance, when you start the *mre* resource editor demo without a filename argument, the program looks for a file called *.Xdefaults* in your home directory. If *mre* cannot find the file, it displays a dialog box containing a message similar to:

```
Couldn't open /home/pat/.Xdefaults.
```

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

Some responses request an action, such as proceeding with a previously invoked process, cancelling the process, or even exiting the program. The dialog box in Figure M-1 contains two push buttons labeled **OK** and **Cancel**. Pushing the **OK** button tells *mwm* to proceed with the restart process. The **Cancel** button gives you a chance to avert the restart process in case you invoked the command by mistake or have changed your mind. Since **Cancel** is highlighted, you can push it either by pressing Return or by using the pointer.

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

As we'll see, some Motif applications support another kind of push button called a *drawn button*. A drawn button is basically a push button decorated with a bitmap rather than text.

The Athena widget set provides comparable widgets to the Motif dialog box and push button. An Athena dialog box provides virtually the same functionality as a Motif dialog. The most obvious difference is that, in an Athena dialog, you must click on a command button to invoke it. The Return key shortcut only works with a Motif push button. See "Dialog Boxes and Command Buttons" in Chapter 9, *Graphics Clients*, for more information about Athena dialogs.

M.1.2 Menu Bars and Pull-down Menus

Figure M-2 illustrates the *menu bar* on a Motifwindow.

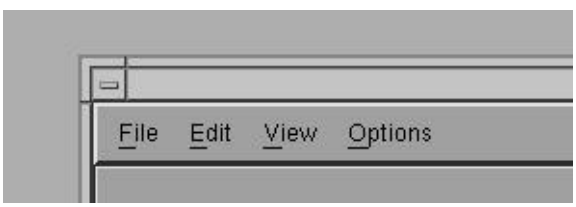


Figure M-2. Motif menu bar

A menu bar is a horizontal bar from which pull-down menus can be displayed. Each word on the bar is a menu title; you display the menu by placing the pointer on its title and clicking the first pointer button. The title becomes raised and highlighted by a box, the menu is displayed and the first selectable item is also raised and boxed. OPEN LOOK applications do not have a menu bar, but have separate main menus, usually in about the same position as a menu bar. One advantage of a menubar is that you can slide the pointer left or right and see all the available menus one after another.

Figure M-3 shows a Motif application's **File** pull-down menu.

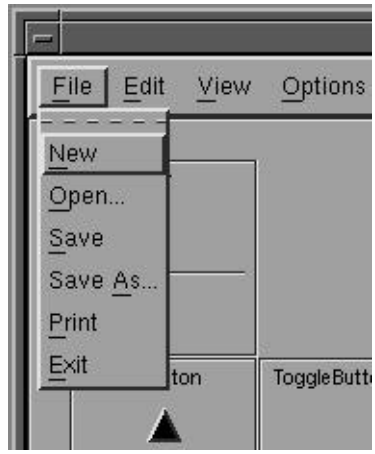


Figure M-3. Motif File menu

Notice that one letter of each menu item is underlined. That letter represents a unique abbreviation, or mnemonic, for the menu item, which can be used in a "mouseless" mode to select the item.

Notice also that each menu item has a keyboard shortcut, or *accelerator*, that appears in the right hand column of the menu. An accelerator can be used to invoke the action without displaying the menu at all (though they also work while the menu is displayed).

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

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

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

M.1.3 File Selection Box

Several Motif applications feature a widget called a *file selection box*, which allows you to select a filename from a list. Most modern OPEN LOOK applications provide similar dialogs; others provide this level of functionality by use of the familiar File Manager (see Chapter 4, *Using the OPEN LOOK File Manager*).

Using a file selection box is not exactly difficult, but it's not particularly obvious either. Let's consider the file selection box that is displayed when you select **Open...** from *mre*'s **File** menu illustrated in the preceding section. The **Open...** menu item is used to read a file into *mre*'s main edit window. When you select **Open...**, a second window is displayed—the file selection box illustrated in Figure M-4.

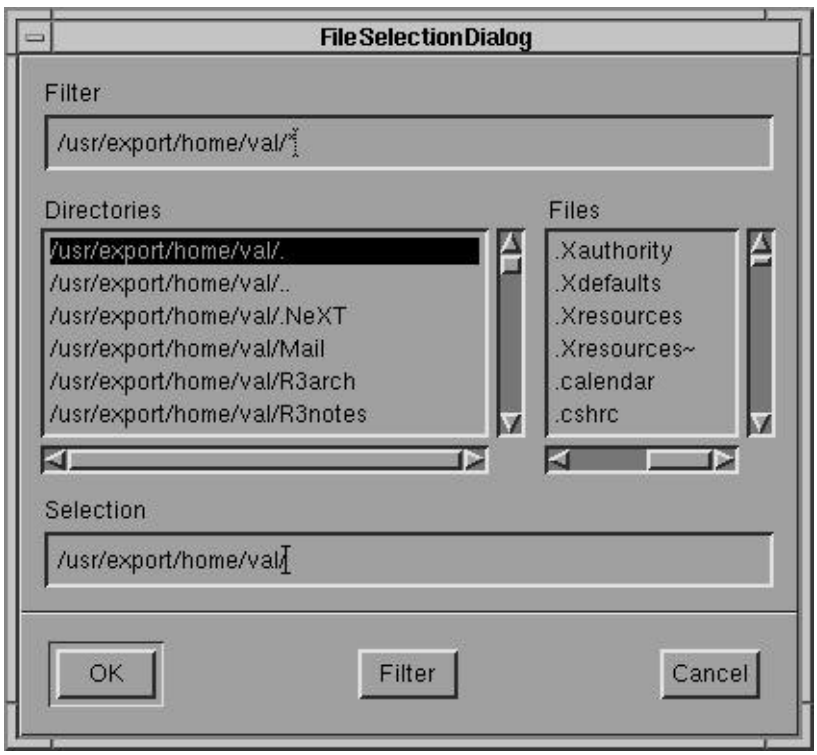


Figure M-4. A file selection box

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

Notice the two areas labeled **Directories** and **Files**. These are *list boxes* that are contained within the larger window. The **Directories** box lists the directories from which you can choose a file; the first directory is usually highlighted. The **Files** box lists the files within the highlighted directory.

Notice that the list boxes are bordered by horizontal and vertical scrollbars, which allow you to view text that is currently outside the box. (The Scrollbar widget is discussed in the next section.) A list box and its accompanying scrollbars form what is known as a Scrolled-

Window. The Motif ScrolledWindow is comparable to the Athena Viewport widget, discussed in Section 8.1.5, "Browsing Reference Pages: xman" on Page 194.

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

M.1.3.1 Selecting a File from the Files Box

To select a file currently in the **Files** box:

1. Place the pointer on the filename.
2. Click the first pointer button. The filename is highlighted by a dark bar; the letters appear in reverse video.
3. Notice also that the **Selection** window will be updated to reflect the filename; and the push button to confirm the selection (**OK** in many applications) will be highlighted, indicating that you can select the file by pressing Return.
4. Select the filename either by pressing Return or by placing the pointer on the **OK** push button and clicking the first pointer button.

When you select a file in *mre*'s file selection box, the file is read in to the initial *mre* editing window and the selection box disappears.

M.1.3.2 Choosing a File from another Directory in the Directories Box

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

1. Place the pointer on the directory name and click the first button. The directory name is highlighted. Notice that the box labeled s-1fHFilterfRs0 is updated to reflect the new pathname and the s-1fHFilterfRs0 push button at the bottom of the box is highlighted for selection.
2. Then, to display the contents of the highlighted directory in the **Files** box either:

- Press Return; or
- Click on the **Filter** push button.

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

M.1.3.3 Choosing a File from Another Directory on the System

You can specify an alternative directory from which a file can be selected by changing the filter, that is, the path in the **Filter** window (near the top of the file selection box). Initially the **Filter** window reflects the current working directory. In Figure M-4, the filter is */work/motif/demos/mre/** and the **Directories** box lists two directories:

```
/work/motif/demos/mre/.\"the current directory
/work/motif/demos/mre/..\"previous directory in the tree
```

To specify another filter, place the pointer within the **Filter** window and double click the first pointer button. The window becomes highlighted with a black bar (the text is visible in reverse video); now whatever you type will replace the current text.

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

```
/home/pat/*
```

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

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

M.1.4 The Motif Scrollbar

Each of the list boxes in the File Selection Box features both a horizontal and a vertical scrollbar. A vertical scrollbar is commonly used review text that has scrolled off the top of a window or extends past the bottom. In the case of *mre*'s **Files** box, the vertical scrollbar is used to scan a list of files too long to fit in the window at one time. A horizontal scrollbar is commonly used to view text or graphics that are too wide to fit in the viewing area. You'll probably encounter vertical scrollbars most often, as in Figure M-5.

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

Now let's take another look at the **Files** box from *mre*'s file selection box, which is bordered by two scrollbars. A Motif scrollbar is comprised of four parts, labeled in Figure M-5: two *arrows* (one at either end of the bar), the *scroll region* between the arrows, and the *thumb*, the raised area which moves within the scroll region. The thumb displays the position and amount of text currently showing in the window relative to the amount saved. If text does not extend beyond the window, the thumb fills the entire scroll region. In Figure M-5, the thumbs in both scrollbars indicate that text extends beyond the bounds of the window.

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

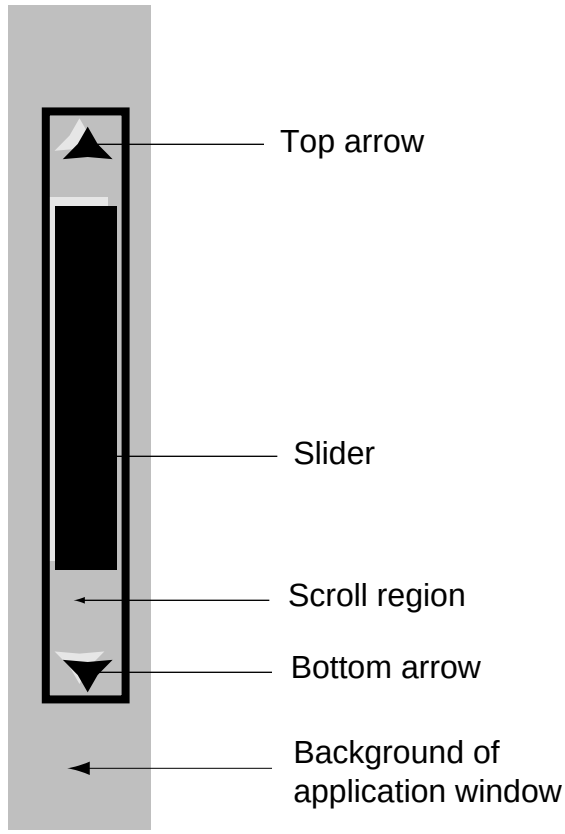


Figure M-5. Motif Scrollbar

of text at the bottom of the window; each click on an up arrow lets you view one more line of text at the top of the window.

A horizontal scrollbar lets you view the remaining part of lines that are too wide to fit in a single window. You use the same pointer commands to use a horizontal scrollbar as you do a vertical scrollbar; obviously the orientation of text and directions of movement are different. Clicking to the right of the thumb scrolls the text horizontally to the right. Clicking to the left of the thumb scrolls the text horizontally to the left. In Figure M-4, the **Files** box is displaying filenames only—the earlier parts of the pathnames are not in view. Notice that the horizontal scrollbar's thumb is all the way to the right of the scroll region. If you place the pointer to the left of the thumb and click the first button, the text is scrolled to the left to reveal the earlier parts of the pathname. Clicking on either arrow of the horizontal scrollbar moves the text one character to the left or right, depending on the direction of the arrow.

The unit scrolled when you click on an arrow depends on the application. Scrollbars are also sometimes featured on application windows that contain graphic elements rather than

text. Obviously, such a window cannot be scrolled by text characters or lines. The *mwm* icon box, described in Chapter 13, *Customizing olwm*, can be scrolled the height or width of one icon.

M.1.5 Drawn Buttons

A *drawn button* is a push button decorated with a pixmap rather than a text label. Figure M-6 shows four drawn buttons from *mre*'s main window.

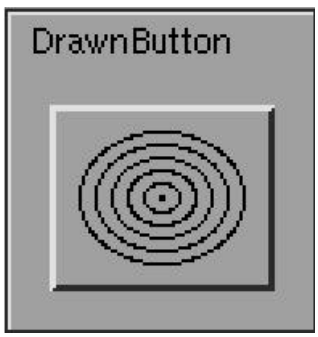


Figure M-6. Motif drawn buttons

The image on a drawn button should signal its function. If a program uses drawn buttons effectively, they can enhance an application's aesthetics.

mre uses drawn buttons well. The button decorated with an artist's palette tells *mre* to place resource specifications relating to color in the application's editing window. The button showing the letter "A" places resources specifying fonts in the editing window. The button featuring the mirror image of the arrows in reverse colors is a rather clever graphical representation of a difficult concept. The image is a sort of technical yin-yang symbol: it tells *mre* to place resources that can be toggled (turned on or off; set to be true or false, yes or no, etc.) in the editing window. Finally, the button featuring the eternity symbol tells *mre* to put all of the resources in the user's resource file in the editing window.

M.1.6 Radio Boxes and Toggle Buttons

A *radio box* is made up of a column of toggles (mutually exclusive choices). Figure M-7 shows four Motif radio boxes in *mre*'s Font Selection window.

Each column is one radio box; each box contains several diamond shaped *toggle buttons*. You push a toggle button by placing the pointer on the diamond symbol and clicking the first pointer button. The toggle button becomes darker (appearing as if it's been pressed). Actually, if you examine the button closely, the highlighting has just switched from the bottom edge to the top edge of the button. When you first make a selection from a column, the button and the accompanying text label are highlighted by a box. When you make a selection in another column, the highlighting box appears in that column (and disappears from the previous one).

Figure M-7. Motif radio boxes

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

M.2 Motif Applications under OLWM

Applications written for the Motif toolkit conform to the vendor-neutral ICCCM document (see Volume Zero, *X Protocol Reference Manual*) so they will operate with any ICCCM-conferment window manager, including *olwm*. It is thus possible to have a "hybrid" environment.

The most common problems we have seen have been related to "OSFkeysyms." If you get a batch of messages about keysyms beginning with the string "OSF" when starting a Motif client, ensure that your Keysyms file (*/usr/openwin/lib/XKeysymDB* or */usr/lib/X11/XKeysymDB*) contains a number of lines beginning with the string "OSF". If not, get a later copy from an X11R5 site or from OpenWindows 3.1.

If you are migrating to Motif, or sometimes use one interface and sometimes the other, you can simplify your life by using X Resources to make one behave more like another. For example, Motif normally uses Alt-F4 to kill an application. You can easily make olwm behave the same way by setting the resource

```
olwm.MenuAccelerator.Quit: F4+Meta
```

in your X defaults file. Similar customizations can be had by perusing Appendix K, *OPEN LOOK Mouseless Operations*

M.3 OPEN LOOK Applications under mwm

M.3.1 X-based clients

Applications written for the X-based OPEN LOOK toolkits generally conform to the vendor-neutral ICCCM document (see Volume Zero, *X Protocol Reference Manual*) so they will operate with any ICCCM-conformant window manager, including *twm* and the Motif window manager *mwm*. It is thus possible to have a "hybrid" environment

However certain features such as pushpins will not always operate. You may need to use the "f.delete" item in your *twm* or *mwm* menu in order to close an OPEN LOOK dialog window. Under *mwm*, you may be able to dismiss pop-up windows by double-clicking the left pointer button on the "menu" icon at the left of the titlebar.

As well, you may find that some Motif-based X servers are deficient in their supply of OPEN LOOK-specific fonts. If you get messages about missing fonts, see Appendix B, *OpenWindows and X11 Fonts*, which talks about adding fonts to the server.

M.3.2 NeWS-based clients

Applications written using the NeWS toolkit, of which there are very few[1], will only operate with an X server that has the "X/News" features; these applications are **not** compatible with a modern "Display PostScript" X server. As well, such applications will have OPEN LOOK-style titlebars and window menus, regardless of what window manager is in operation. It is recommended that you not run these programs under *twm* or *mwm*. Figure M-8, for example, shows the effect of running the NeWS-based OPEN LOOK FrameMaker under *twm* using the OpenWindows 3.0 server. Notice that FrameMaker and the window with this text in it have titlebars and a menu from the OPEN LOOK GUI, while everything else has *twm* titlebars. The same would happen with *mwm* or any other X-based window manager..

1. The only known applications that this applies to are the 3.1 OPEN LOOK version of FrameMaker, the Hyperlook and SimCity programs (which don't use OPEN LOOK conventions anyway), and the Hexsweeper game mentioned in Appendix N, OPEN LOOK Programs.

M

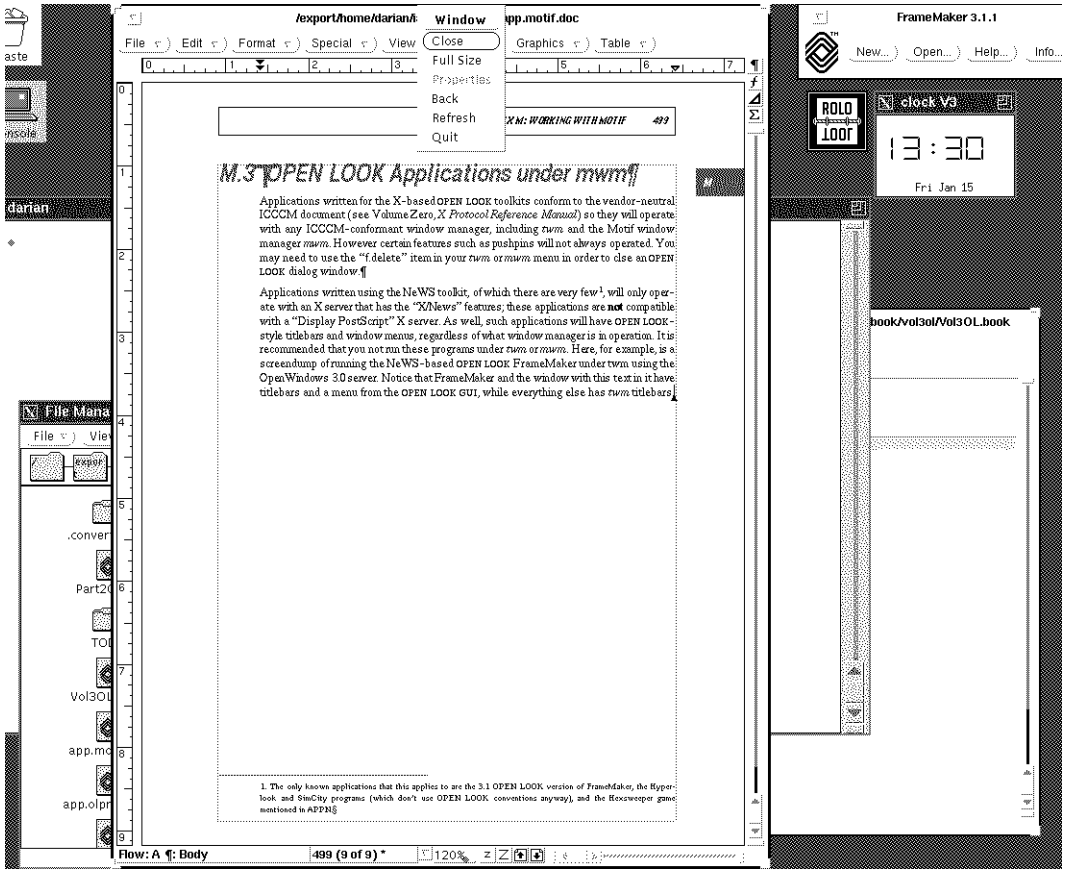


Figure M-8. NeWS Applications running under a non-OPEN LOOK window manager

This page intentionally left blank

to preserve original page counts.

This page intentionally left blank

to preserve original page counts.

APPENDIX N

OPEN LOOK Software Availability

There is a wide range of software available to users of the OPEN LOOK GUI. The following is a reprint of the "Frequently Asked Questions" listing of OPEN LOOK software from the USENET newsgroup *comp.windows.open-look*[1]. It is a compilation of both free and commercial software. This list is updated periodically; the most recent version is normally posted to the Usenet newsgroup *comp.windows.open-look*.

Most of the free software is distributed in source code form, needing compilation.

```
This is version 1.13 92/12/18 of the file open-look-programs.faq.

Contents:
    Subject: Applications: Application Builders
    Subject: Applications: Graphing Tools
    Subject: Applications: Other
    Subject: Tools: Terminal Emulators
    Subject: Other Commercial Applications
    Subject: Applications: toolkit Extensions
    Subject: OpenWindows 3 Ports
    Subject: XView 3 Ports
    Subject: XView 2 Ports
    Subject: Games (free and commercial)
```

N.1 Applications: Application Builders

```
Commercial: DevGuide 3.0
Contact: SunPICS
```

lets you use Drag and Drop to create an OPEN LOOK application with XView, OLIT, UIT or TNT. Very easy to use.

1. This "FAQ" is reprinted with the permission of its compiler, Liam Quin <lee@sq.com>

```
Free: dirt
```

there _might_ be an OLIT port of this UI builder.

```
Commercial: ExoCode
Contact: Expert Object
```

one of the first third-party GUI builders to support OPEN LOOK, using the XView toolkit. It was reviewed in SunExpert magazine in 1990.

```
Commercial: uib
Contact: ParcPlace, Debra Frances debra@ParcPlace.COM, +1 303 678-4626
```

uib is a user interface builder which supports building applications that support both OPEN LOOK and Motif. It generates code for ParcPlace's OI C++ toolkit and can make use of user created subclasses. Note: 'OI' can also display an OSF/Motif GUI at runtime.

```
Free: wcl
```

Uses X resources to specify an Xt widget hierarchy and actions to user-defined callbacks. uses OLIT, Xt or Motif.

```
Commercial: XVT
Contact: XVT Systems (+1 303-443-4223)
```

Lets you write code to a common subset of OPEN LOOK, Motif, Microsoft Windows, the Macintosh GUI, and even terminals (using curses). You buy an XVT toolkit for each environment.

N.2 Applications: Graphing Tools

```
Free: dstool
```

XView-based program that plots Lorenz Attractors and other chaotic things in real time. Also includes a mathematical expression interpreter.

ftp: macomb.tn.cornell.edu

```
Free: ACE/gr -- graph and analysis program, xvgr
Ftp: ftp.ccalmr.ogi.edu [129.95.72.34]; xvgr-2.09.tar.Z in /CCALMR/pub/
acegr
```

Handles x-y scatterplots, lineplots, bargraphs, FFT analysis, running averages, polynomial fits, etc.

```
Free: robot - a scientific graph plotting and data analysis tool
Contact: Robin Corbet <corbet@astro.psu.edu>
Description:
```

Graph plotting in various styles & axes; Data manipulation - arithmetic, functions, smoothing, folding, sorting; Fitting to data using Gaussians, polynomials, Lorentzians, and/or user defined functions; Annotation of graphs; log files; Commands with loops etc.; Colour; PostScript output.

```
Ftp: astrod.astro.psu.edu (128.118.147.28) in pub/astrod
Ftp: files: robotx0.35.tar.Z - everything
```

```
Ftp: files: RobotManual.ps.Z - just the documentation.
Ftp: files: robot.sun4.Z - binary built on a SPARCstation.
Requirements:
```

Robot is XView based. User interface portions of code are written in 'C'. Data manipulation code is written in FORTRAN. Hence a FORTRAN compiler is also required or the public domain f2c package. Alternatively, a SPARC binary is available by anonymous ftp.

N.3 Applications: Other

```
Commercial: Bimail 400
Contact: BIM (+32-2-759.59.25) pge@sunbim.be
X.400-address: C=be;A=RTT;P=BIM;O=Horizon;S=Geurts;G=Patrick
Notes:
```

Bimail is a complete X.400 electronic mail system. It consists in a user interface which gives access to all X.400 services with a consistent look and feel, a message transfer agent (MTA) system which can transfer messages over X.25, TP.4 and TCP/IP (using RFC 1006). A gateway to SMTP mail is also available.

```
Free: WorkMan - Audio CD player for X11 (Sun, Ultrix)
Requirements: XView libraries
Ftp: Ultrix binary: ftp.hyperion.com in /WorkMan
Ftp: ftp.ucsc.edu in "incoming" - database of over 750 CDs
Contact: koreth@hyperion.com (Steven Grimm)
Free: pan - Postit notes
Free: xrolo - Rolodex card index/address book
Commercial: SearchIt 1.0
Contact: SunSoft or SunExpress
SunExpress can be reached at:
```

US: 1-800-873-7869; UK: 0800 89 88 88

Germany: 01 30 81 61 91; France: 05 90 61 57

```
Platforms: SPARC, Solaris 1.x
Price: $249
Notes:
```

SearchIt is a full text search and retrieval application designed to improve individual and group productivity. It makes an index to files and can later retrieve documents by words or phrases, ranking the results in relevance order.

```
Commercial: ShowMe
Contact: SunSoft
Notes:
```

Conferencing software that lets multiple connected users share the same drawing screen, with bitmap capture and moveable pointer.

```
Requirements:
```

You can only run one ShowMe per computer, so you have to have a CPU per conference member.

```
Free: xvman - Man Pages viewer
Free: xvtdl - ToDo List manager
Ftp: export.lcs.mit.edu /contrib/xvtdl-4.0.tar.Z, /contrib/xvtdl-4.0-
README
Requirements: XView libraries
Contact: Mike Jipping jipping@cs.hope.edu (BITNET: JIPPING@HOPE)
Organisation: Hope College Department of Computer Science


Free: name_finder
Contact: richard.elling@eng.auburn.edu +1 (205) 844-2280
Ftp: ftp.eng.auburn.edu [131.204.10.91] pub/name_finder1.2.tar.Z.
Patches: pub/name_finder1.2.compile.patch1.
Requirements: OpenWindows 3.0, C++ 2.1 or greater to recompile
Desctription:
```

name_finder was orginally designed as a replacement for the name finder missing from the OpenWindows Version 3.0 mailtool. It has since grown into a tool for several electronic mail related activities including: interaction with local ListServ robots for handling mail lists, requesting Full.Name style mail aliases from your local Post-Master, and providing mailbox status information ala finger(1). name_finder is written in C++ (cfront 2.1) using gxv++ version 1.1. If you don't have access to a C++ compiler, a precompiled sparc executable is included in the distribution.

```
bibcard (v 1.01) LaTeX Bibliography Manager ???
Ftp: ftp.eunet.ch in software/text/TeX/bibcard-1.0.tar.Z
xftp interface to ftp.
Ftp: ftp.chpc.utexas.edu as file /packages/X/xftp.0.1.alpha.tar.Z.
Contact: Bill Jones jones@chpc.utexas.edu
Requirements: X11, OLIT or Motif or Athena widgets
```

compiles under (at least) Ultrix, AIX 3.1.5, AIX 3.2, Convex OS, SunOS, Unicos 6.1.4, and IRIX. Uses OLIT. BUG: can also use OSF/Motif and Athena widgets.

```
Free: olvwm -- OPEN LOOK Virtual Window Manager
contact: Scott Oaks
Ftp: export.lcs.mit.edu in the contrib directory
Patches: there are two patches
Requirements: XView 3
Description:
```

Olvwm is a version of olwm that manages a 'virtual desktop' (hence the 'v' in its name). It shows a little map on the screen, with the currently displayed area represented by a little rectangle. You can move around by dragging the rectangle or with the arrow keys. This lets you run several clients (applications) and move the display around from one to the other. Olvwm was derived from the OpenWindows 3.0 olwm.

```
Free: bibcard -- OPEN LOOK font end to BibTeX.
Ftp: iam.unibe.ch [130.92.64.10] /X11/Bibcard-1.0.tar.Z
Ftp: includes source and SPARC binary for SunOS 4.1.1.
Requirements: XView
contool (ftp from export.lcs.mit.edu)
```

a special-purpose console-window that can filter out or take special action on specified console messages; written by Chuck Musciano.

`ftptool (ftp from export.lcs.mit.edu)`

an OPEN LOOK front-end to ftp (it uses XView)

`Hyperlook, from the Turing Institute`

Hypertext package written entirely in NeWS. Runtime from turing.com in /pub or ftp.uu.net (graphics/NeWS/HyperLook1.5-runtime.tar.Z)

`Maestro (ftp from sioux.stanford.edu)`

Multimedia authoring tools, including support for sound, text & video.

`xvnews (ftp from export.lcs.mit.edu)`

An xview-based newsreader for netnews.

`xvttool (ftp from titan.rice.edu:sun-source)`

A vt100/102 emulator, in both XView and SunView versions. Includes buttons for the PF keys, etc.

N.4 PostScript and Graphics Viewers

```
Commercial: pageview - PostScript previewer
Contact: Included in OpenWindows as part of DeskSet.
Notes:
```

Type 1 support only in OpenWindows 3.0.1 under Solaris 2.1. Antialiasing support - with colour OpenWindows 3 try pageview -aa -dpi 150 Note that pageview uses the X11/NeWS server to interpret the PostScript, and thus won't run on an X terminal or other non-OpenWindows server. It's *not* enough to be runing an OPEN LOOK UI [tm] window manager such as olwm.

```
Commercial: xps - PostScript program editor and previewer
Contact: included with OpenWindows 2.0 under demo and share/src
Notes:
```

Only runs under OpenWindows 2 (not 3).

```
Commercial: psh
Contact: included with OpenWindows
```

simple interface to NeWS and the OpenWindows server

```
Free: ralpage
Ftp: export.lcs.mit.edu in contrib/clients
Notes:
```

Crispin Goswell's PostScript interpreter, much hacked. Not OPEN LOOK compliant. No Type 1 font support. There are other versions of this called 'xps', 'postscript', etc.; don't confuse this 'xps' with the one mentioned above.

```
Free: ghostscript
(from the Free Software Foundation)
```

Supports Type 1 fonts. Not OPEN LOOK based.

N.5 Tools: Terminal Emulators

```
Free: cmdtool, shelltool
Requirements: XView 3 toolkit
Notes:
```

These are included in the XView source distribution from export.lcs.mit.edu in /contrib; they're also included with Sun's OpenWindows.

```
Commercial: SwitchTerm
Contact: Micro Resources Inc., Columnbus, Ohio, USA, +1 614 766-2335
Notes:
```

A version of Xterm with an OPEN LOOK UI, print interface, ANSI X3.64 colour escape sequences, etc.

```
Commercial: IsoTerm
Contact: The Bristol Group Ltd., +1 415 925-9250 and (49) 6105-2945
(Germany)
Requirements: OpenWindows 3 (??)
Other Products: IsoTeX, IsoFax, Power Base
Notes:
```

An OLIT-based terminal emulator. I couldn't get the demo version to give me a shell prompt, although it did look like it was a pretty fll vt340 emulation, with double-height characters, colour, fonts, grahics and so forth. With the Union Flag (the British flag) as their logo I somehow expected an English address, perhaps in Bristol...

N.6 Other Commercial Applications

Contact SunSoft (or Sun) and ask for the Catalyst OPEN LOOK guide, which lists over 200 pages of applications. You can also get the free CDWare CD/ROM, which contains demo versions of several popular OPEN LOOK UI applications. Once you've done this, you can often simply contact the vendor concerned to have the license upgraded from demo, and receive the full product documentation.

```
Product Name: Author/Editor - SGML-based text editor/word processor
Company Name: SoftQuad Inc., +1 416 239 4801, mail@sq.com
Description:
```

Word processor or text editor that manipulates ISO 8879 SGML documents. Interfaces: OPEN LOOK UI (XView), OSF/Motif, Mac, MS/Windows

N.7 Applications: toolkit Extensions

```
Product Name: Xtra XWidgets
Company Name: Graphical Software Technology
E-Mail: info@gst.com
Phone: 310-328-9338; Fax: 310-376-6224
Keywords: graphics, library, widgets, spreadsheet, help
Interfaces: OPEN LOOK, Motif
```

```
Platforms: SPARC, HP9000s300/400/700, IBM RS6000, Interactive 386
Requirements: X11, Xt, Xol (or Xm) libraries and headers; X11
Price: $795/single user, $3000/network, $5000/source
Support-Price: $400/30 calls
Source-Available: yes
Description:
```

The Xtra XWidget library contains a set of widgets that are subclassed from and compatible with either OLIT or Motif widgets. The library includes widgets that implement the following: Spreadsheet, Bar Graph, Stacked Bar Graph, Line Graph, Pie Chart, XY Plot, Hypertext, Hypertext based Help System, and Data Entry Form. Widgets have been successfully integrated with both TeleUSE from Telesoft and Builder Xcessory from ICS. A free demo is available for any of the supported platforms.

```
Product Name: XRT/Graph
Company Name: KL Group
E-mail: sun.com!suncan!klg!xrt_info
Phone: +1 416 594-1026
Description:
```

XRT/Graph is a charting/graphing extension to XView. There are OLIT and (I _think_) Motif versions available, too. A free demo is available.

```
Free: Slingshot XView extension
```

Slingshot provides rectangles (like the Xt Intrinsics' RectObj gadget), drag-and-drop support, images, icons and text, trees, lines, arrows... Get it by ftp from export.lcs.mit.edu, in /contrib/SlingShot2.0.tar.Z (remember to use binary mode in ftp!).

You can also get it by sending mail to archive-server@gazooch.eng.sun.com with the body of each message containing a line like

send sspkg2.0 Part01 Part02

going up to

send sspkg2.0 Part17 Part18

send sspkg2.0 DocPart01 DocPart02 DocPart03

send sspkg2.0 DocPart04 DocPart05 DocPart06

You can ask for one file at a time to reduce the impact on intermediate mail sites. Ask the mail server for help with the Subject line: "help". A human can be reached at archive-manager@gazooch.eng.sun.com. Add a line in the message

path <your-mail-address>

if you think the normal automatic reply address might not work.

```
Ada bindings for XView
```

Sun Ada 1.1 includes among other things an Ada Source Code Generator for Devguide. It uses the Verdix XView Ada bindings. It does not yet [July 1992] support gfm (the guide file manager).

```
C++ Bindings for XView
Qualix's XV++.
UIT
```

N.8 OpenWindows 3 Ports

Sun: SPARC, SunOS 4.1

Sun: SPARC, Solaris 2 (actually 3.0.1?)

others: none so far...

There are said (by Sun) to be two or three ports of OpenWindows either available now or in progress. Contact Anthony Flynn at Open Vistas International (anthony@ovi.com) for more information. (originally they said 35, but perhaps they meant 3.5)

OpenWindows source is available - commercially, it costs about $5,000 for the server, including TypeScaler and the toolkits; deskset (filemgr etc) is another $25,000; ToolTalk is $40,000 or so.

N.9 XView 3 Ports

```
What: XView 3
System: Apple A/UX
Porter: lmj@uncompaghre.jax.org (Lou Jones)
Ftp: encyclo.jax.org
Notes:
```

The libraries and utilities (olwm, cmdtool, etc) are available for anonymous ftp from encyclo.jax.org. I used gcc 2.1 to compile the sources. If there is enough interest, I can make the diffs available.

```
System: Concurrent 7000 (68040 based)
Porter: sinan@Mtesol.boeing.com (Sinan Karasu)
System: DECStation/Ultrix
Porter: dscott@ittc.wec.com (Dave Scott)
Ftp: media-lab.media.mit.edu:~ftp/xview3-ultrix.4.2-mips.tar.Z
Notes:
```

Let me stress that this is *not* fully tested, but seems to work pretty well. Please let me know about any problems you find. Problems I already know about:

Large buttons under *any* non-Sun X server (non-xnews; i.e. any standard MIT X11R[45] server) have the bottom of the button chopped off. We're working on this one. :-)

```
[actually this seems *not* to be Dave Scott's port; please accept my
apologies for listing this incorrectly. A correct entry will appear as
soon as I get the necessary information. -- Lee]

System: HP 720
Porter: (?)
Ftp: tesla.ucd.ie [137.43.24.44], /pub
```

Notes:

Includes HP 720 build, HP XView patch file, Xvgr.

System: HP9000/300 series
Porter: tjc@ecs.soton.ac.uk (Tim Chown)
System: HP9000/7XX series
Ftp: ftp.csc.liv.ac.uk (138.253.42.172) hpux/X11/xview-3.part[123].tar.Z
System: Intel (SysVR4/i386)
Porter: dawes@physics.su.OZ.AU (David Dawes)
Ftp: ftp.physics.su.oz.au, suphys.physics.su.oz.au /Esix_4/x11r5 hierarchy
Notes:

His patches were for Esix 4.0.3 but should work on DELL, ISC and Intel SVR4 with no worries. The files are README.xview3 and xview3.diff.Z.

System: IBM RS/6000
Porter: tmcconne@sedona.intel.com (Tom McConnell)
Compiler: bsdcc
Ftp: export.lcs.mit.edu:contrib/xview3/Fixes/xview3_rs6k_unofficial.patch.Z
Notes:

There is still a problem with tty support for the RS/6000. For instance, the cmdtool will not work. Still, most everything else works. For those of you who have already installed my previous patch, I have put a separate patch for just the shared library problem. This file is contrib/xview3/Fixes/xview3_rs6k_XView_lib.patch.Z.

System: SGI
Porter: Rainer Sinkwitz <sinkwitz@ifi.unizh.ch>
Ftp: export.lcs.mit.edu:/contrib/xview3/Fixes/xview3_sgi_unofficial.patch.tar.Z
Notes:
System: Solbourne Series 5
Porter: tmcconne@sedona.intel.com (Tom McConnell)

N.10 XView 2 Ports

In general, there is no point in using XView 2 if you have XView 3 available;
it's a good idea to look for an XView 3 port first. Moving from XView 2 to XView 3 is usually simply a matter of recompiling, unless you've done "dirty tricks" or used undocumented calls.
System: Stellar GS100 (Stardent 1000) and Stardent 1500 & 3000
Porter: arvai@scripps.edu (Andy Arvai)
Ftp: perutz.scripps.edu (137.131.152.27) in the pub/xview directory
Notes:

Stardent is now Kubota Pacific (KPC)

System: Harris Nighthawk 4000 system (CX/UX Unix)
Porter: andy@harris.nl (Andy Warner)

```
Status: Commercial
System: SGI/Iris
Porter: (?)
Ftp: wuarchive.wustl.edu:graphics/graphics/sgi-stuff/XView/xview2
System: VAX/VMS
Porter: TGV Inc (?)
Notes:
```

Steven Fenger <svfenge@afterlife.ncsc.mil> wrote:

A company called TGV makes a product called "XView for VMS". ...

N.11 Games (free and commercial)

```
Free: hexsweeper - minesweeper game based on hexagons
Contact: lee@sq.com, include HexSweeper in Subject
Requirements: OpenWindows 3.0 or later
Toolkit: TNT 3
Free: Spider (Included in OpenWindows under 'demo' and 'share/src')
```

A patience-style card game with two packs of cards and excellent bitmap cards. I suggest recompiling to allow the cards to have rounded edges.

```
Free: Xblackjack (ftp from export.lcs.mi.edu as contrib/xblackjack-
2.1.tar.Z)
```

A MOTIF/OLIT based tool constructed to get you ready for the casino.

```
Commercial: Aviator - flight simulator for GX-equipped SPARCStations
Contact: Artificial Horizons Inc, aviator-interest@ahi.com; +1 415 367
5029
Requirements: OpenWindows (2 or 3), SunOS 4.1 or later, SPARC GX or
GXplus
Commercial: SimCity
Contact: Dux Software, Los Altos, CA
Price: US$89
Requirements: OpenWindows 3 (uses NeWS). Doesn't run on a 4/110 with cg4
:-(
```

Glossary

X and OPEN LOOK use many common terms in unique ways. Good examples are "server" and "child." While most, if not all, of these terms are defined where they are first used in this book, you will find it easier to refresh your memory by looking here.

The following defines the meanings of most of the common X and OPEN LOOK terms used in this book. Terms defined elsewhere in the glossary appear in **bold.** Terms that are specific to OPEN LOOK, as well as those to which OPEN LOOK assigns special meanings, are identified with the tag (OL) after the term.

abbreviated menu (OL)

A menu **button** displayed as a small square with an inverted triangle inside it. Most commonly seen in OPEN LOOK titlebars. Behaves like an OPEN LOOK menu button.

abbreviated scrollbar (OL)

A **scrollbar** on a very small **panel** or **split pane** may be displayed with just the **cable anchors** and **arrow buttons**

accelerator

A key or key sequence that is short for one or more menu and/or button actions.

access control list

X maintains lists of hosts that are allowed create windows on a display. By default, only the local host may use the display, plus any hosts specified in the *access control list* for that display. Nowadays it is recommended to use **X Authorization**. instead.

active window	The window where the input is directed. With a ``pointer focus" policy, such as *olwm* in pointer-focus mode, or *twm*, you must put the pointer in a window to make it the active window. The active window is sometimes called the f**ocus window.**
ADJUST pointer button (OL)	The pointer button, normally the middle one on a three-button mouse, used to adjust— lengthen or shorten— a selection.
application	See under client.
arrow button (OL)	An abbreviated button, usually found at the ends of a text field, used for scrolling. A special case of "abbreviated button."
ASCII	American Standard Code for Information Interchange. This standard for data transmission assigns individual 7-bit codes to represent each of a specific set of 128 numerals, letters, and control characters. See also **iso8859-1.**
Athena	A project at the Massachusetts Institute of Technology; in X11, commonly used to refer to the Athena Widget Set used with the **X Intrinsics Toolkit.**
Automatic Scrolling (OL)	Automatic scrolling lets you move the view by holding SELECT and wiping through (moving the pointer across) the data. At present, this works for OLIT clients, but not for XView clients.
background	Windows may have a *background*, consisting of either a solid color or a tile pattern. If a window has a background, it will be repainted automatically by the server whenever there is an **Expose** event on the window.
background color	The color that determines the backdrop of a window, for example, on monochrome displays, the root window background color is gray.
background window	Another name for the **Root Window** or **Workspace**.
Backing Store	An attribute of the X Server that lets it retain in memory an image of overlapped areas so that the **client** doesn't have to redraw them.

base frame (OL), base window

The main **window** of an application. A base frame has a close button (shown here) at the left of its title bar; a pop-up window generally has a **push-pin**. See also **Command Frame, Properties Window.**

binding

An association between a function and a key and/or pointer button. The OPEN LOOK File Manager *bind*s file types with actions, so that clicking on the file's icon causes it to be processed in some predefined way; see Chapter 4, *Using the OPEN LOOK File Manager*. The MIT window manager `twm` allows you to *bind* its functions to any key(s) on the keyboard, or to a combination of keys and pointer button (e.g., the Control key and the middle button on a 3-button pointer).

bitmap

A grid of pixels or picture elements, each of which is white, black, or, in the case of color displays, a color. The OpenWindows DeskSet client *iconedit* and the MIT *bitmap* client allow you to edit *bitmap*s, which you can use as pointers, icons, and background window patterns.

border

OPEN LOOK windows have a very thin *border* around them; you can access the **window menu** from any point on the border. With non-OPEN LOOK **window manager**s, a window can have a *border* that is zero or more pixels wide. If a window has a *border*, the *border* can have a solid color or a tile pattern.

button

A tiny ``window" area with a label that, when clicked on, causes some action to be performed. In OPEN LOOK, buttons can be used to perform an action directly, to display a pop-up window, or to display a menu.

Cable (OL)

The Cable is the part of a **scrollbar** that shows how much data can be scrolled, and lets you scroll by one screenful at a time.

Cable anchor (OL)

The boxes at the end of the cable anchor; clicking SELECT on them scrolls all the way to the appropriate end of the data. Dragging SELECT on them lets you split the view.

Cancel — A button that lets you remove a property sheet or other pop-up dialog without applying any changes; the action of so removing it. Also a key that does so, usually <ESC> or STOP.

Caret (OL) — A small mark that shows you which of several text fields will receive your keyboard input. The active *caret* is a black half-diamond; the inactive *caret* is a grey diamond.

check boxes (OL) —  A nonexclusive choice setting; shows a check mark or tick mark in a small box when the setting is active, and is an empty box otherwise.

click — To press one of the **pointer button**s to notify a program of your intentions; the action of so pressing.

client — X11's name for an application program, any program that requests the X Server to create and manage one or more **window**s for it. There are standard *client* programs to perform a variety of tasks, including terminal emulation and window management. Commercial or third-party applications are considered to be **client**s. **Client**s need not run on the same system as the display server program**.**

clipboard — A holding area into which you can paste selected text or other matter.

close — To close a window is to iconify it. On OPEN LOOK**,** this does not terminate the client controlling the window; on OSF/Motif and on Microsoft Windows, closing a window means to quit from it.

colorcell — An entry in a colormap is known as a *colorcell.* An entry contains three values specifying red, green, and blue intensities. See also **colormap**.

colormap — A **colormap** consists of an array of colorcells. A pixel value indexes into the colormap to produce intensities of red, green, and blue to be displayed. The colormap on most systems is a limited resource that should be conserved by allocating read-only colorcells whenever possible, and selecting RGB values from the predefined color database. Read-only cells may be shared between clients. See also **RGB**.

colormap lock (OL)

An OPEN LOOK facility to lock the colormap of one application onto the display, to avoid flashing of colors when moving the pointer in and out of that window. On the OpenWindows version of olwm, the key sequence Control/L2 locks, and Control/L4 unlocks, the colormap of the window that has the input focus.

console window

This **window**, which usually contains a terminal emulator such as *cmdtool*, *contool*, or *xterm*, is usually the first window to appear on your display. On some implementations, exiting the console window kills the X **server** program and any running applications. Also called the **login terminal window.**

control

Any screen area used to activate a function. Examples include menus, scrollbars, buttons, etc.

There are also *control keys* on most computer keyboards; these are often used as **accelerators**

copy cursor (OL)

A cursor indication that the drag-and-drop selection is being copied, rather than moved.

cut

To cut a selection (graphics or text) is to remove it from the active window; it is usually stored in a clipboard so it can be pasted in elsewhere.

default

A function-dependent value assigned when you do not specify a value. For example, specifying the **-rv** option with *xterm* reverses the foreground and background colors for the *xterm* window. If you do not specify this option, the **default** foreground and background colors are used.

default ring (OL)

In an OPEN LOOK pop-up, the *default* button is indicated by a double line around it. In an OPEN LOOK menu, the default item is indicated by a thin oval ring around it. You can change the default in a menu by dragging with Control-ADJUST.

Desktop manager

Any program similar to OPEN LOOK'S File Manager.

depth

The *depth* of a window or pixmap is the number of bits per pixel.

device-dependent	Aspects of a system that vary depending on the hardware. For example, the number of colors available on the screen (or whether color is available at all) is a *device-dependent* feature of X.
DeskSet (OW)	A set of **clients** provided with Sun's OpenWindows that provides some useful functionality that is either missing or costs extra with most other GUIs. See Chapter 7, *The OpenWindows DeskSet Clients*
Dialog box	A **popup** window used to ask for specific information.
display	A set of one or more screens driven by a single X **server**. The DISPLAY environment variable tells programs which servers to connect to, unless it is overridden by the `-display` command line option. The default is always screen 0 of display server 0 on the local system. See Chapter 1, *An Introduction to OPEN LOOK and the X Window System*
Display PostScript	An extension to X11 that allows rendering of PostScript files into X windows. Not the same as Sun's earlier **NeWS** protocol for PostScript.
double-click	To click twice in rapid succession on a control. This is normally a shortcut or **accelerator** for selecting something and then clicking another button to activate it.
drag and drop	A protocol brought to the X11 community by OPEN LOOK that lets you copy a selection from one window to another with a single mouse action: you hold a mouse button down while dragging the pointer, and release the button to "drop" the selection in another window.
Drag Area (OL)	The area in the middle of a scrollbar or slider that is dragged to scroll the data or change the slider's value.
Drop Target (OL)	A rectangle into which you can drop a selection to make it current; you can often drag from the drag target to get the current contents of the application.
Drop Shadow (OL)	The gray area to the right and below the border of a menu.

Duplicate pointer (OL) — The cursor displayed when you are copying an object by dragging.

Elevator (OL) — The central feature of an OPEN LOOK scrollbar; it features up and down arrows to scroll up or down, and a central drag area that can be moved by dragging.

event — The window **server** represents user actions such as pointer motion, key presses and window **exposure** with a data structure called an *event*. Application programs interpret these *event*s and respond to them in particular ways.

exposure — Window *exposure* occurs when a window is first put up on the screen, or when another window that obscures it is unmapped (e.g., iconified or destroyed), resized, or moved. Expose events are sent to clients to inform them when contents of regions of windows have been lost and need to be regenerated.

Exclusive choice (OL) — A **control** that allows for only one of a number of choices to be selected at one time.

Extension — The MIT X **server** allows a skilled programmer to add new features; a set of one or more such features is called an *extension*.

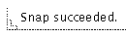
File Manager (OL) — A program (*filemgr*) used to browse through files and directories. It displays different icons for different types of files, and can automatically start application programs or take other action when you double-click on files. See Chapter 4, *Using the OPEN LOOK File Manager*, for a discussion of the OPEN LOOK file manager.

focus window — The window to which keyboard input is directed. For a discussion of the two modes of selecting the *focus window*, see Chapter 2, *Working in the OPEN LOOK Environment*

focus help (OL) — A visual indication of which window has the input focus. Normally invoked by Control-?.

font — A style of text characters. Fonts and X font naming conventions are described in Chapter 10, *X11, OPEN LOOK and OpenWindows Font Specification*.

font directory — By default, fonts are stored in the subdirectories of */usr/lib/X11/fonts* (MIT) or $OPENWIN-HOME*/lib/fonts* (OpenWindows). You can specify alternative or additional *font directories* with the *xset* client.



footer (OL) — The bottom area of a main window, used to display messages. The left footer is for status messages such as the completion of an action; the right footer is for information such as the current page number.

foreground color — The color in which the text in windows and menus, or graphics output are displayed.

Front (OL) — The Front item in the Window Menu moves a window to the front of the screen, that is, over top of any windows that overlap it. In OpenWindows the accelerator for this is the L5 key.

Full Size (OL) — To make a window *full size* is to make it use the largest size (normally by height) that it can. This is a Window Menu item and is also available by double-clicking on a window's titlebar.

gauge (OL) — An output-only control that shows the percentage of use or the proportion of an action that has been completed.

geometry — The size and placement of a window, which can be specified with the `-geometry` option or an X resource.

Graphical User Interface (GUI) — A specifications for the look and feel of client windows. OPEN LOOK is one GUI; the MacIntosh Operating System and the OSF/Motif interface are others.

header (OL)

The OPEN LOOK term for a titlebar. A small rectangle across the top of the window, with the program's name or other information (for example, the name of the file being processed in the window) centered. Base windows have a window menu button (**menu mark**) at the left of the header; pop-up windows (command and property windows) have a pushpin at the left.

Help

Assistance or information about a window or control. OPEN LOOK applications provide help in a consistent way, by the use of a HELP key (F1 on some keyboards; HELP on others).

hexadecimal

A base-16 arithmetic system, which uses the digits A through F to represent the base-10 numbers 10 through 15. X clients accept a special hexadecimal notation (prefixed by a # character) in all command line options relating to color. See XX CHAPTER WITH COLOR XX for more information.

highlight

As you drag the pointer down a menu, each item is *highlighted* by drawing it in a different color. Releasing the pointer button on the highlighted item will cause it to be selected.

hot spot

The reference point of a pointer that corresponds to its specified position on the display. In the case of an arrow, an appropriate *hot spot* is its tip. In the case of a cross, an appropriate hot spot might be its center.

icon

A small symbol that represents a window but uses little space on the display. Converting windows to *icons*, called "iconifying", allows you to keep your display uncluttered. X11 and NeWS programs typically have an icon that is intended to be representative of the program. Because they have the full capability of PostScript available, NeWS programs often have a scaled version of the current image of the main window.

input device — Hardware device that allows you to input information to the system. For a window-based system, a keyboard and pointer are the most common input devices. The X Input **Extension** allows you to add new device types to the MIT X server.

internationalization — A way of writing programs that allows use of a different human language than that used by the author of the program.

iso8859-1 — A character set containing **ASCII** and a number of international (European) characters as well.

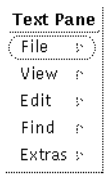
join views (OL) — To join back together two views of a panel; see *split view*.

keyboard accelerator — See accelerator.

keyboard focus — See **focus window**.

keyboard input — The entry of information from the keyboard; the text entered from the keyboard.

maximize — The Motif term for *Full Size.*



menu — A list of commands or functions, listed in a small window, one of which can be selected with the pointer.

menu button (OL) — A button that has a menu associated with it. An example is the File menu on most main windows; clicking SELECT will take the default action (load or save as appropriate), while clicking MENU will cause the menu to be displayed. See Chapter 2, *Working in the OPEN LOOK Environment*

menu mark (OL) — A triangle in the border of a button or to the right of a menu item, to indicate that a menu is connected to it. Triangle points down or right to show which direction the menu will appear in. See also **Abbreviated Menu Button.**

	MENU pointer button (OL)	The pointer button, usually the rightmost one on a three button mouse, that is used to make a menu appear.
	Minus button (OpenWindows only)	A button used in some of the OpenWindows DeskSet clients to reduce the size of the Properties window.
	modifier keys	Keys on the keyboard such as Control, Alt, and Shift. X programs recognize a set of "logical" *modifier key* functions that can be mapped to physical keys. The most frequently used of these logical keys is called the "meta" key) labelled ¨Left¨ and ¨Right¨ on some keyboards=.
	MoOLIT	An AT&T toolkit that will be able to provide either an OPEN LOOK or a Motif interface, at the user's request. Derived from their earlier OLIT.
	Motif	A Graphical User Interface and its toolkit, developed by the Open Software Foundation by coalescing elements of DEC's and Hewlett-Packard's X-based GUIs.
	mouse	An input device that, when moved across a flat surface, moves the pointer symbol correspondingly across the display. The mouse usually has buttons that can be pressed to send signals that in turn accomplish certain functions. The mouse is one type of pointer device; the representation of the mouse on the screen is also called the **pointer**. (See **pointer**.)
	Move cursor (OL)	A pointer shape used to warn you that a drag you are doing will move, rather than copy, the data object. Moving a file from the File Manager to the Print Manager, for example, will print the file and delete it from your disk.
	Multimedia	A system that has access to media other than the traditional screen/keyboard/mouse, such as sound, interactive video, etc.

NeWS	Sun's Network Extensible Window System, an alternative to X11. Unlike Display PostScript, NeWS extends the PostScript language into a full windowing system. The OpenWindows server supports both X11 and NeWS clients concurrently.
NFS	The Networked File System, allows files from almost any operating system to be "mounted" on any other computer that is connected via a network.
non-exclusive choice (OL)	A control that allows for any or all of its multiple choices to be active (selected) at one time.
Notice (OL)	A Notice is OPEN LOOK's term for a **pop-up** window used to bring your attention to an error or warning that requires action, usually a choice, before the application can continue. Analogous to certain types of **dialog box** in other terminologies., and to the ¨Alert Box¨ on the MacIntosh.
occluding	In a windowing system, windows may be stacked on top of each other much like a deck of cards. The window that overlays another window *occludes*, or hide, that window. A window need not completely conceal another window to be occluding it.
OLIT	The OPEN LOOK Intrinsics Toolkit, an OPEN LOOK tookit from AT&T.
open	To open a window is to replace its icon with the full-scale window.
OPEN	The most-abused term in the computer industry; the marketing buzzword of the 1980's and 1990's. It began with Sun's use of "Open Systems", but has since lost much of its meaning through overuse.
OPEN LOOK	A Graphical User Interface developed by Sun and AT&T; the first industry-standard GUI for the X11 and UNIX environments. The trademark in the name is held by AT&T. Oops, better make that Novell, Inc.
OpenWindows	Sun's windowing software package; includes an X server, most standard X clients, window and file managers, a series of useful DeskSet clients., the ToolTalk inter-application protocol, X toolkits, etc.

OSF/Motif	See under Motif.
page-oriented scrollbar (OL)	A scrollbar that moves from one page to the next in a page-oriented application such as publishing software. Has a small box to indicate the current page number.
palette	A set of colors used to select the colors for parts of your display.
parameter	A value required before a client can perform a function. Also called an argument.
paste	To insert a selection that was previously cut or copied.
picture frame corners	See Resize Corners.
Pinnable menus (OL)	Menus that have a pushpin and can thus be pinned up on your screen until you are done with them.
pixel	The smallest element of a display surface that can be addressed.
Plus button (OpenWindows only)	A button used in some of the OpenWindows DeskSet clients to increase the size of the Properties window.
point	A unit of measurement equal to approximately 1/72 of an inch (0.354 am).
pointer	A generic name for an input device that, when moved across a flat surface, moves the pointer symbol correspondingly across the display. A *pointer* usually has buttons that can be pressed to send signals that in turn accomplish certain functions. The **mouse** is by far the most common type of pointer device at present. The *pointer* also refers to the symbol on your display that tracks pointer movement on your desk. Pointers allow you to make selections in menus, size and position windows and icons, and select the window where you want to focus input. A pointer can be represented by a variety of symbols. (See **text cursor**.) Some typical X pointer symbols are the I-beam and the skull and crossbones.

pop-up menu (OL)	A menu that pops up when you press the MENU key anywhere other than over a **menu button**.
pop-up window	A window that appears in order to perform some specific tasks. Examples of pop-up windows in OPEN LOOK include command windows, property windows, help windows, and notices.
property (X11)	Windows have associated *properties*, each consisting of a name, a type, a data format, and some data. These are used for many purposes in X11, such as specifying whether a window is to be iconified, what Resource values are to be used, etc.
Property (OL)	OPEN LOOK uses the term *properties* in a more conventional sense than X11´s technical meaning, to mean attributes of something, such as the type of font to use, or the color of a window.
Property Window (OL)	A pop-up window used to set preferences ("properties") of an object, a program, or a window.
 Pushpin (OL)	A graphic image of a pushpin used to make a window or menu stay up until you explicitly dismiss it by removing the pin.
Pin	See pushpin.
quit	To quit an application means to terminate it. The OPEN LOOK Window Manager has a quit item in its window menu, so you can quit any application from the window manager.
radio button	A group of buttons where only one of the buttons can be selected at a time. So named because they behave like the channel buttons of an old fashioned car radio. OPEN LOOK's analog is the **exclusive choice**.
reverse video	Reversing the default foreground and background colors.
RGB	A method for defining color in which proportions of the primary colors Red, Green, and Blue are combined to form other colors.
Refresh	To make the window (or the entire screen) be redisplayed.

Resize corners (OL)

Most windows can be made smaller or larger by selecting one of their resize corners and sliding it in the desired direction to make the window larger or smaller.

Root Menu

The menu that a window manager displays when you request a menu with the pointer positioned on the background, that is, not inside any given window. OPEN LOOK calls this the **Workspace Menu**.

Root Window

The background area of your screen; the area on top of which all actual windows are drawn. Also called the **background window** or, in OPEN LOOK, the **Workspace**.

RPC

Remote Procedure Call, a high-level communication method used by some applications to exchange information. NFS uses RPC.

screen

A server may provide several independent *screens*, which may or may not have physically independent monitors. For instance, it is sometimes possible to treat a color monitor as if it were two screens, one color and one black and white.

scroll

To move through data that is larger than the window that is displaying it.

scrollbar

A bar on the side of a window that allows you to use the pointer to scroll up and down through the text saved in the window. Useful whenever the amount of data in an application is greater than amount that can be displayed at one time.

scrollbar menu (OL)

Each OPEN LOOK scrollbar has a menu that can be used to scroll, and sometimes to **split/join** the view of the window. Access the menu by pressing MENU with the pointer over the scrollbar.

select
: A process in which you move the pointer to the desired menu item or window and click or hold down a pointer button in order to select some text or data.

SELECT pointer button (OL)
: The pointer button, usually the leftmost one on a three-button mouse, used to start a selection or to select a particular choice from a menu or series of buttons.

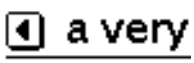
selection
: *Selections* are a means of communication between clients using properties and events. A selection is an object or item of data that can be highlighted in one instance of an application and dragged or pasted into another instance of the same or a different application. The current selection can also be copied into the **clipboard**, or **cut**.

server
: A server is a program that makes some resource available to clients. A traditional "file server" makes file storage available to workstations. In X terminology, the server is the combination of graphics display, hardware, and X server software that provides display services for clients. The *server* also handles keyboard and pointer input.

split view (OL)
: To divide a panel into multiple views of the data; each view has its own scrollbar and can be scrolled independently.

stay-up menus (OL)
: Menus that stay on the screen when you click and release the MENU button.

Text Arrow (OL)
: Visual indication that a text field is larger than can be displayed in the visible portion on the screen.



text cursor
: The standard underscore or block cursor that appears on the command line or in a text editor running an *xterm* window. To make the distinction clearer, the cursor that tracks the movement of a mouse or other pointing device is referred to as the **pointer**. The pointer may be associated with any number of cursor shapes, and may change shape as it moves from window to window.

text duplicate pointer (OL)	Pointer that displays when a text drag is copying the data
text move pointer (OL)	Pointer that displays when a text drag is (destructively) moving the data.
tile	A pattern that is replicated (as if laying a *tile*) to form the background of a window or other area. This term is also used to refer to a style of non OPEN LOOK window manager that places windows side by side instead of allowing them to overlap.
titlebar	A small rectangle on top of a window. Shows the name of the application, and allows menu access to various window manager functions.
tNt, TNT	The NeWS Toolkit, a **toolkit** for developing **NeWS** applications under **OpenWindows.**
toolkit	A set of programmer-accessible functions used to develop applications. Examples of OPEN LOOK toolkits for X11 include AT&T's OLIT and Sun's XView, and for NeWS, Sun´s TNT. There is only one Motif toolkit, the Motif Toolkit. There are several OPEN LOOK- and Motif-compliant toolkits from third parties.
ToolTalk	A Sun program that provides very high level interprocess communication. Used by some of the **deskset** clients to exchange messages; used by *filemgr* to start some clients.
triple-click	To **click** three times in rapid succession. Normally used to select an entire line of text in an editor or terminal emulator.
unbundled	Sold separately from the base system; costing extra money.
Undo	To undo an operation, i.e., to return the data to its state before the last change you made.
Unicode	An extension of the **ASCII** character set designed to include all known European and Asian languages See also **iso8859-1**.

	UNIX	A multi-tasking, multi-user operating system that is most commonly used on mini-computers and work-stations. The X Window System was developed on UNIX, but now runs on several other operating systems or even stand-alone in **X Terminals**.
	Validate	To verify that an item of data is acceptable in a given context.
	Virtual Desktop	A scrolling facility for making the **Workspace** appear larger than the physical screen. Examples of window managers that implement this include *tvtwm* and *olvwm*.
	Virtual Edges (OL)	A facility for reserving part of the workspace for icons and certain application windows. Rarely used.
	Virtual Keyboard	An image of the keyboard used to show and alter the mappings of various keys.
	Virtual Reality	Nothing to do with X11; a fancy multimedia system that appears to present you with an interactive, three-dimensional world.
	Wait Cursor	A cursor indicating that you've asked the application to do something that may take a long time. On some GUIs the watch hands appear to move; this does not happen with most current X11 implementations.
	window	A region on your display created by a **client**. For example, the *cmdtool* or *xterm* terminal emulator, the *calctool* or *xcalc* calculator, and the *iconedit* or *bitmap* graphics editor all create windows. You can manipulate windows on your display using a **window manager**.
	window manager	A client that allows you to move, arrange, resize, circulate, and iconify **windows** on your display. The *Window Manager* also draws borders and titlebars. The most common *Window Manager*s under OPEN LOOK are *olwm* and *olvwm*.
...	window mark	Three dots (...) following a button label or menu item to indicate that it will cause a separate pop-up window to appear.

window menu (OL)	A menu that is displayed when you press the MENU button on a window's titlebar.
Wipe-through selection	Pressing SELECT and moving the pointer across data (text or graphics) to make a continuous selection.
Workspace (OL)	OPEN LOOK'S term for the **Root Window**, the region behind all other windows
Workspace Menu (OL)	OPEN LOOK'S term for the **Root Menu**, the menu that 9zis accessed when the MENU button is pushed over the Workspace (not over any window).
Workspace Properties Menu (OL)	The property window accessed from the Workspace Menu that lets you customize global properties by updating your *.Xdefaults* file. See Chapter 2, *Working in the OPEN LOOK Environment*.
X Authorization	A mechanism that allows your programs to open windows on your display, regardless of what host they are running on, without having to be explicitly added to the Access Control List. The most common form of X Authorization uses the file *~/.Xauthority* to contain an encryption that the server recognizes. See the manual pages for *xauth* and *xhost*.
X Terminal	A computer terminal that is dedicated to running The X Window System
X11	A window system that is the subject of this series of books. The X Window System is a trademark of the Massachusetts Institute of Technology.
Xt+	An early name for the **OLIT** toolkit.
XView	Sun's OPEN LOOK toolkit for X11. Has a similar programmer's interface to their older, pre-X11 SunView toolkit.

Documentation Roadmap and Bibliography

This is a bibliography of works on the OPEN LOOK GUI, OpenWindows, and X11. Some sections of this bibliography are adapted from material in the Usenet newsgroup *comp.windows.open-look* Frequently Asked Questions (FAQ).

Other Books in the O'Reilly X Window System Guides series

The O'Reilly X Window System Guides provide a comprehensive set of documents for The X Window System. Other books in the series which may be of interest to end-users include:

X User Tools. ISBN 1-56592-019-8. Linda Mui, Valerie Quercia, and other authors (including the author of the present work). A new (1994) work that provides the most comprehensive coverage imaginable of all the programs for The X Window System, including standard X11 programs and free software. Includes one chapter on the OPEN LOOK GUI.

X Window System Administror's Guide (Volume 8), ISBN 0-937175-83-8, with CD: 1-56592-052-X). This volume discusses many aspects of setting up and running the X Window System, particularly in a multi-vendor environment.

X Window System in a Nutshell. (Nutshell series) Contains reference pages for many of the standard clients, along with programmer documentation on the system.

Most of the remaining books in the series are of interest to programmers developing and maintaining X Window System applications:

Volume Zero, *X Protocol Reference Manual*, discusses the network protocol used between the X server and its clients. Of interest mainly to advanced programmers.

Volumes and Two, *XLib Programmer's Manual* and *Xlib Reference Manual* discuss the use of XLib, the lower level of access.

Volumes Four and Five, X Toolkit Intrinsics Programmer's Manual. and *X Toolkit Intrinsics Reference Manual*, discuss the details of the "Xt" toolkit library.

Volume Seven, *XView Programming Manual*, discusses Sun's XView toolkit for building OPEN LOOK applications.

Volume Six, *Motif Programming Manual*, discusses in full detail the Motif toolkit.

PHIGS Programmer's Manual. (X series, no volume number) discusses in considerable detail the use of PHIGS graphics and PEX, the Phigs Extension to X, a graphics layer that can be used on top of X.

Power Programming with RPC. (Nutshell Handbook series) discusses developing networked applications using Sun's Remote Procedure Call library, including significant discussion of developing RPC applications that work within the constraints of X toolkits such as Sun's OPEN LOOK toolkit XView and Xt (including OLIT and Motif).

Books about OPEN LOOK

The official description and usage guidelines for the OPEN LOOK GUI are contained in the following two books:

- Sun Microsystems Inc., *OPEN LOOK Graphical User Interface Functional Specification*, Addison Wesley
- Sun Microsystems Inc., *OPEN LOOK Graphical User Interface Application Style Guidelines*, Addison Wesley, 1989

Writing Programs for OPEN LOOK

David Miller describes programming with OLIT in *An OPEN LOOK At Unix* (M&T press).

Nabajyoti Brkakati gives an excellent introduction to X and to OLIT programming, as well as setting up and using X and OpenWindows, in *Unix Desktop Guide to OPEN LOOK*, SAMS, 1992 ISBN 0-672-30023-0 You can get the examples from this book as *ftp://export.lcs.mit.edu/contrib/naba-olguide-examples.tar.Z*

Also about using OLIT, and Xt in particular: *The X Window System: Programming and Applications with Xt*, OPEN LOOK Edition, Doug Young and John Pew, Prentice Hall, 1992, ISBN 0-13-982992-X. There are also HP Widgets and Motif versions of this book. The example source code in this book can be obtained as f*tp://export.lcs.mit.edu, file contrib/young.pew.olit.Z.*

There is an introduction to XView in *Writing Applications For Sun Systems*, Vol 1, *A Guide for Macintosh(R) Programmers* (Sun Microsystems, pub. Addison Wesley)

A recent (the last?) XView programming guide is *Practical XView Programming*, by Kenneth W. Bibb and Larry Wake, John Wiley & Sons, Inc., 1993, ISBN 0-471-57460-0You can get the examples from this book as *ftp://export.lcs.mit.edu:/contrib/xvprac.tar.Z*

NeWS, PostScript and Display PostScript

To learn more about the NeWS and PostScript languages, see

The NeWS Book, Springer Verlag, 1989 (sadly, a little out of date)

Sun's documentation on NeWS, accompanying OpenWindows releases up to 3.2. OpenWindows versions up to version 3.2 use NeWS which is a level 1 PostScript implementation, with certain Level 2 features (such as Composite Fonts) to some degree. Versions 3.3 and higher feature Level 2 Display PostScript.

PostScript Language Reference Manual, Second Edition, Adobe Systems Inc., Addison Wesley, 1990. ISBN 0-201-18127-4 ("the Red Book"). Note: the first edition of this book, ISBN 0-201-10174-2) is sometimes still available, but it is substantially out of date.The first edition describes only Level 1 PostScript; the second edition also describes Level 2 PostScript.

PostScript Language Tutorial and Cookbook, Adobe Systems Inc., Addison Wesley, 1985, ISBN 0-201-10179-3. ("The blue book"). A fairly gentle introduction to PostScript.

P*rogramming the Display PostScript System with X*. Adobe Systems Inc., Addison Wesley, 1993, ISBN 0-201-62203-3. Contains several books in one: Programming Guide, Client Library Reference and Supplement for X, pswrap Reference, and DPS Toolkit for X. Comprehensive if nothing else.

Sun Documentation Roadmap

Sun also supplies a large amount of documentation with OpenWindows, although you may have to order it separately. Here are some that came with one version; note that the part numbers are subject to change. Most of these also appear in the Answerbook, and some are also available in bookstores.

800-6006-10 OpenWindows Version 3 Release Manual

800-6029-10 OpenWindows Version 3 Installation and Start-Up Guide

800-6231-10 OpenWindows Version 3 DeskSet Reference Guide

800-6618-10 OpenWindows Version 3 User's Guide

800-6323-10 Desktop Integration Guide

800-6027-10 Programmer's Guide

800-6005-10 OpenWindows Version 3 Reference Manual [the man pages]

800-6055-10 OLIT 3.0 Widget Set Reference Manual

800-6198-10 XView 3.0 Reference Manual: Converting SunView Applications

There are also some other sets of documentation, including the TypeScaler documentation from Sun's OpenFonts group, for example.

Books about ToolTalk

ToolTalk is Sun's inter-application communication protocol. It assumed extra significance when it was chosen as the standard inter-application protocol for the COSE vendors' Common DeskTop Environment (CDE). You can read about it in:

ToolTalk 1.x Setup and Administration Guide (SunSoft, 1991)

ToolTalk 1.x Programmer's Guide (SunSoft, 1991)

Sun's AnswerBook

Sun's AnswerBook CD/ROM (see Chapter 7, *The OpenWindows DeskSet Clients*) contains a lot of the above documentation, in PostScript form, and a viewer program to read it online or print parts of it on any PostScript printer. There are several generations of Answerbook software; the earlier ones of course use NeWS and the later ones use Display PostScript.

Frequently Asked Questions (FAQ) lists

This frequently-asked-questions (FAQ) list is a good source for up-to-date information. The FAQ on OPEN LOOK is posted every so often to the Usenet group *comp.windows.open-look* (among others). It can also be obtained by sending electronic mail to lee@sq.com to ask for it. Douglas N. Arnold (dna@math.psu.edu) keeps an up-to-date copy on *ftp.math.psu.edu* in the file ~ftp/pub/FAQ/open-look

There is a World Wide Web version, maintained by Andrew Violette, at *http://cs.indiana.edu/faq/OpenLook/front_page.html*

The FAQ on X itself is posted regularly to the Usenet group *comp.windows.x*.

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Colophon

This volume is derived in part from the Standard and Motif editions of this book, which were produced with SoftQuad *troff*, with most of the figures produced in Aldus Freehand on a Macintosh. The Alpha Draft of the OPEN LOOK edition was produced in the same manner, although many of the Macintosh-produced figures were replaced with screendumps of OPEN LOOK applications. The text was converted to FrameMaker format with a series of scripts written by Dale Dougherty and updated by Ian Darwin, and then revised for OpenWindows Version 3. The second draft and most revisions were produced using FrameMaker on a Sun SPARCstation. The remaining drawn figures are mostly produced in Aldus Freehand by Chris Reilly.

About the authors

Ian Darwin

Ian is the author of *Checking C Programs with lint* and over fifty articles, courses and seminars on the UNIX operating system and related software. When not working for SoftQuad Inc., producers of software for HTML/SGML and corporate publishing, he teaches UNIX-related courses for Learning Group International, writes for *Sun Expert* magazine and, of course, O'Reilly & Associates. Ian lives north of Toronto with his wife and their three children.

Valerie Quercia

Valerie is a staff writer for O'Reilly & Associates and co-author (with Tim O'Reilly) of several editions of *Volume 3: X Window System User's Guide*. Her personal trainer and mnager, Dr. Heinrich Bunsen, reports: "Though X has been very, very good to her, Val would really like to try another letter, maybe a nice Q."

Val lives close enough to Boston to be upset by the Red Sox.

Tim O'Reilly

Tim is well known in the UNIX and X communities as the founder of O'Reilly & Associates, publishers of high-quality, low-cost UNIX and X documentation.He wrote the earliest ancestral version of this book.