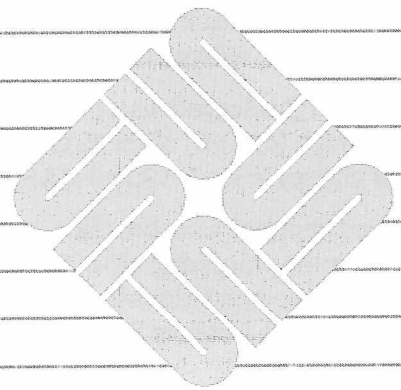





X11/NeWS 1.0 Server Guide



NeWS[™], X11/NeWS[™], SunView[™], XView[™], OpenFonts[™], F3[™], and OpenWindows[™] are trademarks of Sun Microsystems, Inc. Sun Workstation[®], Sun Microsystems[®], and the Sun logo [®] are registered trademarks of Sun Microsystems Inc.

POSTSCRIPT[®] is a registered trademark of Adobe Systems Inc. Adobe owns copyrights related to the POSTSCRIPT language and the POSTSCRIPT interpreter. The trademark POSTSCRIPT is used herein to refer to the material supplied by Adobe or to programs written in the POSTSCRIPT language as defined by Adobe.

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

UNIX[®] is a registered trademark of AT&T.
OPEN LOOK[™] is a trademark of AT&T.

Times[™], Helvetica[™], and Palatino[™] are trademarks of the Linotype AG and/or its subsidiaries.

Bembo[™], Gill Sans[™], and Rockwell[™] are trademarks of Monotype Ltd.

ITC Avant Garde[™], ITC Bookman[™], ITC Zapf Chancery[™], and ITC Zapf Dingbats[™] are trademarks of International Typeface Corporation.

Lucida[®] is a registered trademark of Bigelow and Holmes.

LaserWriter[™] is a trademark of Apple Corporation.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations.

Copyright © 1989 Sun Microsystems, Inc. – Printed in U.S.A.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means – graphic, electronic, or mechanical – including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

Restricted rights legend: use, duplication, or disclosure by the U.S. government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

The Sun Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees.

Contents

Chapter 1 Introduction to the X11/NeWS Server	3
1.1. The Server Start-up Procedure	4
1.2. The XView and NeWS Toolkits	4
1.3. Definitions of Terms	5
window	5
server-based window system	5
kernel-based window system	5
window server	5
client	5
window management	5
1.4. Applications	5
Chapter 2 Using the NeWS Protocol	9
2.1. NeWS Utilities	9
2.2. NeWS Initialization Files	10
The init.ps File	10
The .startup.ps File	11
The .user.ps File	11
2.3. Modifying .startup.ps	11
Changing the Initial Screen Image	11
2.4. Modifying .user.ps	12
Conserving Dictionary Space	12
Modifying the Access List	13
Modifying the Root Menu	13

The buildmenu(1) Utility	14
Using the Default Menu File	14
Replacing the Default Menu File	14
Modifying the Default Menu File	14
Abbreviations	14
Changing the Root Background	15
2.5. The UserProfile Dictionary	15
2.6. Running NeWS Programs	17
Executing POSTSCRIPT Files	17
Previewing POSTSCRIPT Graphics	18
Communicating Directly with the Server	18
Connecting to Remote NeWS Servers	19
Running Remote Clients	20
2.7. Compiling Source Programs	21
2.8. Running a NeWS-Only Server	21
Chapter 3 Using the X11 Protocol	25
3.1. X11 Utilities	25
3.2. X11 Features Not Supported	26
3.3. Displaying POSTSCRIPT Images from X11 Clients	26
Using CPS	27
An Example Program	29
3.4. Connecting to Remote X11 Servers	29
3.5. Running Remote Clients	30
3.6. Restricted Access to the X11/NeWS Server	30
3.7. Running an X11-Only Server	30
3.8. X11 Idioms to Avoid	31
Black and White	31
GXset and GXclear	31
Pixmap Contents	31
Visuals	31
3.9. Compiling Source Programs	31

Chapter 4 Window Management	35
4.1. External Window Management	35
4.2. NeWS Window Management	35
4.3. X11 Window Management	36
Using Redirection	37
Example	37
Example	37
The <code>override-redirect</code> Attribute	38
4.4. Window Management in X11/NeWS	38
4.5. Switching Window Managers	38
X11 Window Managers and NeWS Windows	39
Restarting <i>pswm</i>	39
Starting With Other Window Managers	39
4.6. Problems with External Window Management	40
 Chapter 5 Font Support	 45
5.1. Server-Supplied Fonts	45
Locating Fonts	45
The Core Set of Outline Fonts	46
Times	46
Helvetica	46
Helvetica Narrow	46
Symbol	46
Courier	46
New Century Schoolbook	47
Palatino	47
ITC Bookman	47
ITC Avant Garde	47
ITC Zapf Dingbats	48
ITC Zapf Chancery	48
Lucida Family	48
Bembo	48
Rockwell	48

Gill Sans	48
The Minimum Set of Fonts for the Server	49
5.2. Font Families	49
5.3. Accessing and Scaling Fonts	49
5.4. Using Other Fonts	50
5.5. Adding Fonts	51
Adding F3 Fonts	51
Generating Bitmap Files	51
5.6. Converting NeWS 1.1 Fonts for Use with the X11/NeWS Server	52
5.7. Font Aliases	52
5.8. Fonts in the X11 Window System	52
Using OpenFonts with the X11 Protocol	52
X11 Access to NeWS Fonts	53
5.9. Font Limitations	53
Chapter 6 Color Support	57
Color in the NeWS Window System	57
Color in the X11 Window System	57
6.1. X11/NeWS Visuals and Colormaps	57
X11/NeWS Visuals	58
6.2. Recommendations	59
X11 Programmers	59
NeWS Programmers	60
6.3. NeWS Dynamic Colors	60
Chapter 7 Using SunView Windows	63
Running SunView Applications	63
Bugs in SunView/X11/NeWS Coexistence	64
Inconveniences	64
Screen Damage	64
Input Mismatches	64
Appendix A NeWS Manual Pages	65

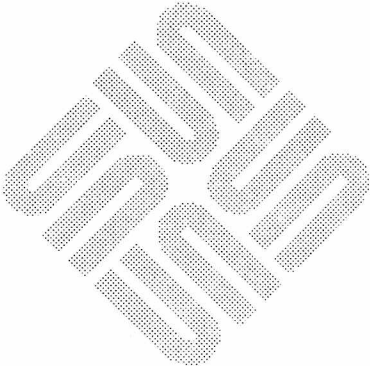
24to8.1	67
bldfamily.1	69
buildmenu.1	71
convertfont.1	73
cps.1	75
journalling.1	77
kbd_mode.1	79
ldf.1	81
makeafb.1	83
mkiconfont.1	85
newshost.1	89
newsserverstr.1	91
objectdiff.1	93
objectwatcher.1	95
openwindemos.6	97
pageview.1	103
pam.1	105
psh.1	107
psindent.1	109
psio.3	111
psman.1	117
psps.1	119
pstags.1	121
xnews.1	127
Appendix B X11 Manual Pages	133
bitmap.n	135
ico.n	143
logo.n	145
maze.n	147
muncher.n	149
plaid.n	151
pswm.n	153

puzzle.n	155
worm.n	157
xcalc.n	159
xdpr.n	163
xdpyinfo.n	165
xfd.n	167
xhost.n	169
xlsfonts.n	171
xlswins.n	173
xmac.n	175
xmag.n	177
xmodmap.n	179
xpr.n	183
xprop.n	185
xrdb.n	189
xrefresh.n	193
xset.n	195
xsetroot.n	197
xterm.n	199
xwd.n	209
xwininfo.n	211
xwud.n	213
Index	215

Tables

Table 2-1 UserProfile Entries 16

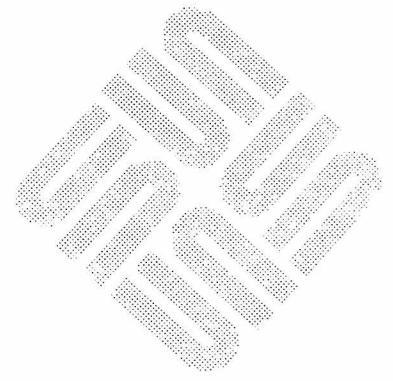
Table 5-1 Minimum Set of Fonts 49





Figures

Figure 5-1 The Core Set of Outline Fonts 47



Preface

This manual describes the X11/NeWS[™] server and its components. The X11/NeWS server is a program that runs on machines suitable for high-resolution graphics and exchanges messages with X11 and NeWS[™] applications, allowing them to display windows on the screen. It forms the window system platform for the OpenWindows[™] environment.

The *X11/NeWS Server Guide* is intended for X11 and NeWS application programmers who are building window tools (for example, text editors or mail tools) to run as part of the OpenWindows user environment. This manual is not a programming guide for the X11 or NeWS protocols. Rather, it is a description of how to run X11 and NeWS programs with the X11/NeWS server.

The X11/NeWS server is based on an early version of the OPEN LOOK[™] Graphical User Interface, and therefore does not necessarily implement every element in the most recent revision of the OPEN LOOK specification.¹ OPEN LOOK references in this manual thus point to software that has not yet been validated by AT&T as fully compliant with OPEN LOOK, although full validation is expected shortly.

Summary of Contents

Chapter 1, *Introduction to the X11/NeWS Server*, contains general information about the X11/NeWS server, including definitions of terms and a list of applications you can run with the server.

Chapter 2, *Using the NeWS Protocol*, contains an overview of the NeWS protocol, and a discussion of how to run NeWS programs with the X11/NeWS server.

Chapter 3, *Using the X11 Protocol*, contains an overview of Version 11 of the X Window System[™] (X11), and discussions of its features and utilities.²

Chapter 4, *Window Management*, describes both NeWS and X11 window management facilities.

Chapter 5, *Font Support*, describes the core set of fonts provided with the X11/NeWS server, and how to use and add fonts.

Chapter 6, *Color Support*, describes the differences between the color-handling techniques for the NeWS and X11 protocols.

¹ OPEN LOOK is a trademark of AT&T.

² The X Window System is a trademark of the Massachusetts Institute of Technology.

Chapter 7, *Using SunView Windows*, describes how to run SunView™ binaries on the X11/NeWS server.

Appendix A contains NeWS reference manual pages, and Appendix B contains X11 reference manual pages.

For More Information

For information on X11 programming, see:

- *Xlib Programming Manual*, O'Reilly & Associates, Inc., 1988
- *Xlib Reference Manual*, O'Reilly & Associates, Inc., 1988

For information on NeWS programming, see:

- *NeWS Programmer's Guide*

For information about the OpenWindows environment, see:

- *OpenWindows User's Guide*
- *OpenWindows Installation and Start-Up Guide*

For information about the POSTSCRIPT® language³, see:

- Adobe Systems Inc., *POSTSCRIPT Language Tutorial and Cookbook*, Addison-Wesley, 1985
- Adobe Systems Inc., *POSTSCRIPT Language Reference Manual*, Addison-Wesley, 1985

Notational Conventions

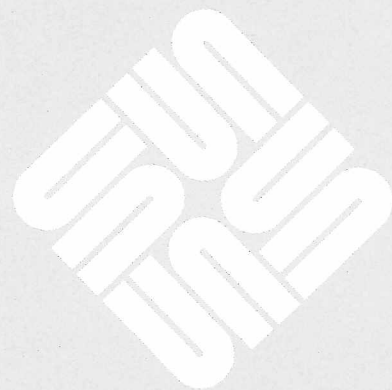
This manual uses the following notational conventions:

- **bold listing font**
This font indicates text or code typed at the keyboard.
- `listing font`
This font indicates information displayed by the computer and the use of the C programming language.
- **sans serif font**
This font is used in code examples to indicate use of the POSTSCRIPT language or NeWS extensions.
- **bold font**
This font is used in textual passages to indicate names of system-defined POSTSCRIPT language or NeWS objects, and to introduce important terms.
- *italic font*
This font indicates user-specified parameters for insertion into programs or command lines. It is also used to indicate special terms or phrases.

³ POSTSCRIPT is a registered trademark of Adobe Systems Inc.

Introduction to the X11/NeWS Server

Introduction to the X11/NeWS Server	3
1.1. The Server Start-up Procedure	4
1.2. The XView and NeWS Toolkits	4
1.3. Definitions of Terms	5
window	5
server-based window system	5
kernel-based window system	5
window server	5
client	5
window management	5
1.4. Applications	5



Introduction to the X11/NeWS Server

The X11/NeWS server is a program that runs on a machine suitable for high-resolution graphics. It acts as a window server, providing display capabilities and keeping track of user input. Application programs — known as *clients* — send messages to the X11/NeWS server and cause it to render images on the display. The images rendered for the clients appear in windows on the screen. The client programs associated with each window may reside on the same machine as the X11/NeWS server or on another machine on the network.

The X11/NeWS server implements the client/server model of window systems, unlike SunView, which is a kernel-based window system. In the client/server model, client programs connect to a window server. The clients send requests to the server to manipulate windows, display images, and communicate with other clients. The server sends information about input devices, window manipulation, and communication from other clients back to the client programs.

The X11/NeWS server represents a merge of the X11 and NeWS window systems — it can handle messages from clients that follow either the X11 or the NeWS protocol. The X11 window system is version 11 of the X Window System developed at MIT. It is a *statically extensible* window system (that is, server suppliers may offer extensions, but clients cannot download their own extensions). It uses a pixel-based imaging model, in which images are viewed as rectangular areas of device-dependent pixels. (See the *Xlib Reference Manual* for more information about X11.¹)

NeWS (Network extensible Window System), developed by Sun Microsystems, is a window system based on the POSTSCRIPT page description language. It is a *dynamically extensible* window system (that is, clients can define their own functions and download them to the server). It is based on a stencil/paint imaging model, in which images are defined in device-independent user coordinates. (See the *NeWS Programmer's Guide* for more information about NeWS.)

¹ *Xlib Reference Manual*, O'Reilly & Associates, Inc., 1988

1.1. The Server Start-up Procedure

You start up the X11/NEWS server by invoking the program `xnews`. You can invoke `xnews` after you log in to your workstation or you can set up your `.login` file to start `xnews` automatically. The server locates and executes `init.ps`, which loads a variety of `.ps` files to initialize the server. Among the `.ps` files are two user-specific files: `~/startup.ps`, which sets server-wide defaults early in the initialization process, and `~/user.ps`, which sets user customizations at the end of initialization. The *OpenWindows Installation and Start-Up Guide* discusses these initialization files in more detail. See also Chapter 2, *Using the NeWS Protocol*, in this manual.

As part of its start-up procedure, the server runs a shellsript named `openwin-init`. The default shellsript starts a File Manager and a `cmdtool`. (See the *OpenWindows User's Guide* for information about these applications.) You can override the default shellsript to start other applications. You can also start other applications (clients of the server) from the `cmdtool` window or from the root menu. The server also initializes the root menu from a file called `openwin-menu`, which the user can also override.

The X11/NEWS server runs as a single UNIX[®] process that listens for new connections to clients.² If a client is running on the same machine as the server, it communicates with the server using streams-based IPC. If a client is running on another machine on the network, it communicates with the server using a network protocol.

1.2. The XView and NeWS Toolkits

The X11/NEWS server forms the window system platform for the OpenWindows user environment. In addition to the platform, OpenWindows contains the XView toolkit and the NeWS toolkit, and a set of client applications based on these toolkits. Since the toolkits support the OPEN LOOK[™] User Interface Functional Specification, the applications provided with OpenWindows and any customer applications built on the toolkits also support OPEN LOOK.³

The XView toolkit provides an object-oriented library of routines and a framework for developing X11 applications. An XView application uses procedure calls to access XView facilities from a C program. (See the *XView Reference Manual* for information about the XView toolkit.)

The NeWS toolkit provides a set of NeWS development components on which application programs and tools may be implemented. The NeWS toolkit consists of a set of server components, written in the NeWS-extended POSTSCRIPT language, and a set of client components, written in C. The server components include OpenWindows user interface components, such as windows, panes, and buttons, and structural components for organizing the user interface components. The client components provide facilities for communicating between the client and server portions of the application.

² UNIX is a registered trademark of AT&T.

³ OPEN LOOK is a trademark of AT&T.

1.3. Definitions of Terms

This section describes some of the important terms used throughout this manual.

window

A *window* is an area of your terminal screen that displays the output of a client. You can display several windows on your screen simultaneously. You can manipulate and interact with each window on your screen independently of the other windows on the screen.

server-based window system

A *server-based window system* centralizes screen access in one UNIX process, the *window server*. Client programs communicate with the window server using a reliable byte stream and therefore do not need to reside on the same machine. They can display output in windows on another machine on the network, regardless of machine architecture, operating systems, display resolutions, or color capabilities.

kernel-based window system

In a *kernel-based window system*, such as SunWindows, coordination of access to the screen is done using extensions to the UNIX kernel, and each client directly accesses the screen.

window server

A *window server*, such as the X11/NeWS server, is a program that runs on a user's machine and handles the display capabilities of the machine. Clients of the window server, which may run on the same machine as the server or on another machine on the network, connect to the server and provide instructions for rendering images. The window server creates and manages windows on the screen for displaying clients' images. The window server also collects input from the user and sends it to clients. Clients process the user's input and send instructions to the server for updating their windows.

client

A *client* is an application program that sends requests to a window server over a reliable byte stream to display and manipulate windows and graphics, and to receive input. A client can run on the same machine as the window server or on another machine on the network.

window management

Window management is a set of functions with which a user can control the layout and state of windows on the screen. The agent that implements these functions is known as the *window manager*. The functions include moving, resizing, opening, closing, raising, lowering, and quitting windows.

1.4. Applications

You can run the following kinds of application with the X11/NeWS server:

- All X11 applications (for example, those built with the XView user interface package)
- NeWS applications that conform to NeWS 1.1 documented interfaces (including Lite toolkit-based applications), or that are based on the NeWS toolkit
- SunView- and SunWindows-based applications

Using the NeWS Protocol

Using the NeWS Protocol	9
2.1. NeWS Utilities	9
2.2. NeWS Initialization Files	10
The init.ps File	10
The .startup.ps File	11
The .user.ps File	11
2.3. Modifying .startup.ps	11
Changing the Initial Screen Image	11
2.4. Modifying .user.ps	12
Conserving Dictionary Space	12
Modifying the Access List	13
Modifying the Root Menu	13
The buildmenu(1) Utility	14
Using the Default Menu File	14
Replacing the Default Menu File	14
Modifying the Default Menu File	14
Abbreviations	14
Changing the Root Background	15
2.5. The UserProfile Dictionary	15
2.6. Running NeWS Programs	17
Executing POSTSCRIPT Files	17
Previewing POSTSCRIPT Graphics	18
Communicating Directly with the Server	18



Connecting to Remote NeWS Servers	19
Running Remote Clients	20
2.7. Compiling Source Programs	21
2.8. Running a NeWS-Only Server	21

Using the NeWS Protocol

NeWS is an interpreted programming language based on the POSTSCRIPT programming language.⁴ The POSTSCRIPT language was developed at Adobe Systems and is used primarily for specifying the format and design of printed documents. A POSTSCRIPT program consists of operations that are sent to a POSTSCRIPT interpreter residing within a printer; when interpreted, the operations define text, graphics, and page coordinates.

NeWS uses POSTSCRIPT operators to display text and graphics. Programs are interpreted and executed by the X11/NeWS server, which is resident on the user's graphics console. Importantly, NeWS also provides operators and types that are extensions to the POSTSCRIPT language; many of these extensions deal with the interactive aspects of window management that the POSTSCRIPT language does not consider.

In addition to these operator and type extensions, which are "hard-wired" into the server, NeWS also contains various POSTSCRIPT language files that provide support for the NeWS programming environment; the files are loaded automatically when the X11/NeWS server is initialized. By modifying the files and the procedures they contain, you can customize the way in which initialization occurs.

This chapter lists the standard NeWS utilities provided with the X11/NeWS server and describes the server initialization process, giving examples of how you can specify your own initialization routine by customizing POSTSCRIPT files. It also discusses the different ways in which you can run NeWS programs.

This chapter assumes you have a basic understanding of the POSTSCRIPT language. A complete description of both NeWS and the names and contents of POSTSCRIPT initialization files is provided in the *NeWS Programmer's Guide*. Information on installation and start-up commands is provided in the *OpenWindows Installation and Start-Up Guide*.

2.1. NeWS Utilities

The following list contains the standard NeWS utilities provided with the X11/NeWS server. See the *NeWS Programmer's Guide* and the reference manual pages in Appendix A for more information.

- `bldfamily`: a facility for building font family descriptions.

⁴ Adobe Systems Inc., *POSTSCRIPT Language Reference Manual*, Addison-Wesley, 1985.

- `cps`: a facility for constructing a C-to-POSTSCRIPT interface.
- `convertfont`: a facility for converting fonts from one format to another.
- `journaling`: a NeWS event record-and-playback package.
- `kbd_mode`: a facility for changing the translation mode of the keyboard.
- `mkiconfont`: a facility for making an ASCII cursor or icon font from a list of ASCII bitmap files.
- `newshost`: a facility for controlling NeWS network security.
- `newserverstr`: a facility for generating a string for the `NEWSERVER` environment variable.
- `objectdiff`: a facility for listing differences between two lists of X11/NeWS data objects.
- `objectwatcher`: a facility for listing data objects allocated and de-allocated.
- `pageview`: a POSTSCRIPT previewer for NeWS.
- `pam`: a facility for removing “stuck” windows from the NeWS display.
- `psh`: the NeWS POSTSCRIPT shell.
- `psindent`: a facility for formatting POSTSCRIPT language or NeWS source.
- `psio`: a NeWS buffered input/output package.
- `psload`: a facility for displaying the load average under NeWS.
- `psman`: a facility for displaying and finding reference manual pages.
- `pmps`: a NeWS process lister.
- `pstags`: a facility for creating a POSTSCRIPT language or NeWS tags file for use with `vi`.
- `psterm`: a NeWS terminal emulator.

2.2. NeWS Initialization Files

This section describes the sequence by which X11/NeWS initialization files are loaded.

The `init.ps` File

When you start up the X11/NeWS server, the server process immediately searches for a file named `NEWS/init.ps`, looking in each of the following directories in turn:

1. `./` (the directory in which the server has been started)
2. `~/` (the user’s home directory)
3. `$OPENWINHOME/etc/`

When the file is found, it is executed. The file contains ASCII POSTSCRIPT code that is responsible for loading most of the other POSTSCRIPT initialization files provided with the X11/NeWS server.

You can modify `init.ps`, thereby creating your own version of the initialization process. The modified version of the file should be placed in the directory `./NeWS` (the first directory searched by the server process) so that no other version of `init.ps` is searched for. Note, however, that the recommended procedure for customizing your initialization procedure is to leave `init.ps` unchanged and create files named `.startup.ps` and `.user.ps`, which can be used to override specifications contained in the `init.ps` file. These user-created files are described in the following sections.

The `.startup.ps` File

One of the first tasks that `init.ps` performs is to search for a file named `.startup.ps` and, if the file is found, execute it. Note that this file exists only if it is created by the user. The search procedure used by `init.ps` is identical to that used by the server process to search for `init.ps` itself.

The `.startup.ps` file is intended to contain ASCII POSTSCRIPT code specified by the user. Typically, such code is used to install new operators or modify existing ones. Note, however, that since the system's own POSTSCRIPT files have not yet been read in when `.startup.ps` is loaded, system-supplied routines that are defined by packages such as `windows` and `cursors` cannot be used in this file. Commonly, the `.startup.ps` file is used to change the initialization screen image and class variable defaults.

After loading `.startup.ps`, `init.ps` loads a standard set of POSTSCRIPT files that define the classes, packages, and user interface for NeWS. (These files and the operators they contain are described in the *NeWS Programmer's Guide*.) It then starts up the NeWS and X11 interpreters. At this point the server can respond to client program requests to display and manipulate windows created with the X11 or NeWS protocols.

NOTE *The `systemdict` contains a dictionary named `UserProfile`, which can be modified in order to customize packages loaded by the server. The modifications to `UserProfile` should be specified in the `.startup.ps` file. See the section The `UserProfile` Dictionary for details.*

The `.user.ps` File

After loading all other files, `init.ps` searches for a file named `.user.ps`. Like `.startup.ps`, `.user.ps` exists only if it has been created by the user. The search procedure used to locate this file is identical to that used for both `.startup.ps` and `.init.ps`. The `.user.ps` file is intended to contain ASCII POSTSCRIPT code that can be used to override default settings and modify or create operators; thus, its role is almost identical to that of `.startup.ps`. However, `.user.ps` can use or reference any NeWS operator or type, since all have been loaded by the time `.user.ps` is itself executed.

2.3. Modifying `.startup.ps`

This section demonstrates how the file `.startup.ps` might be modified.

Changing the Initial Screen Image

You can customize the appearance of the `OpenWindows` root window by modifying `.startup.ps`. The root window's appearance is specified by a procedure named `InitPaintRoot`, which is executed by the server immediately after the contents of `.startup.ps` have been executed. (Note that the default version of `InitPaintRoot` produces the "OpenWindows Version 1.0" message that

appears when you first start up the OpenWindows environment.)

You can redefine `InitPaintRoot` within your `.startup.ps` file, thus producing a root window with a different appearance. The following example specifies that the screen be painted red:

```
/InitPaintRoot {  
  1 0 0 setrgbcolor clippath fill  
} store
```

The image created by `InitPaintRoot` remains on the screen until the server is fully initialized, at which time the X11 root background is painted. The default background is blue, on color screens, and is a gray stipple pattern, on monochrome screens. You can set this background to be a different color, pattern, or image by using the `xsetroot(1)` command. If you run the server in `NEWS-only` mode, or if you set the `UsePaintRoot?` flag in `UserProfile`, then X11 will not paint the root background. Instead, the `PaintRoot` procedure is invoked to paint the root background. `PaintRoot` is executed whenever any part of the root window is exposed. The following example, if added to your `.startup.ps` file, redefines `PaintRoot` to paint the root canvas green:

```
/PaintRoot {  
  0 1 0 setrgbcolor clippath fill  
} store
```

See the section *Changing the Root Background* for a simpler way to change the color of the root window.

NOTE *Since many system utilities have not been loaded when `.startup.ps` is read, you can use only the lowest level of POSTSCRIPT rendering operations in your `.startup.ps` file.*

The recommended way to change the root color is to use the OpenWindows Workspace Properties window.

2.4. Modifying `.user.ps`

This section suggests POSTSCRIPT modifications that you may wish to make to your `.user.ps` file.

Conserving Dictionary Space

Any procedures that you define in your `.user.ps` file will be added by `NEWS` to the *system dictionary*, named `systemdict`. This dictionary has a finite amount of room available and is shared by all `NEWS` processes. Therefore, it is desirable to limit the number of entries within this dictionary and make sure that the new and pre-existing entry-names do not conflict. To do this, you can create your own dictionary from within `.user.ps`; the new dictionary can contain all your extensions, thereby adding only a single entry to the system dictionary itself. This is demonstrated by the following example:


```

systemdict /myVDIdict known not {
  systemdict /myVDIdict 50 dict put
  myVDIdict begin
  /VDIrange 34200 def
  etc.
  end
} if

```

This example checks whether anything named `VDIdict` is already defined in the system dictionary; if nothing is found, a new dictionary of that name is created, adding a single entry to the system dictionary.

You can access the new dictionary's contents as follows:

```

myVDIdict begin
VDIrange 4 mul
etc.
end

```

Alternatively, you can use the `get`, `store`, and `put` primitives.

Modifying the Access List

NEWS can access the control list of all hosts permitted to establish connections with the server. The list is kept in the `RemoteHostRegistry` dictionary in `systemdict` and can be accessed like any other POSTSCRIPT dictionary. The access control variable `NetSecurityWanted` is initialized to true, and `RemoteHostRegistry` contains entries for the local host and all its aliases. If you want to allow constant access to a particular machine, add the following POSTSCRIPT code to your `.user.ps` file:

```

RemoteHostRegistry begin
  /neighbor true def
end

```

This code enables all connections emanating from a machine named "neighbor". You can also change the variable `NetSecurityWanted` in your `.user.ps` file to enable or disable all network access. For example, to allow access to the server from any remote machine, use the following code:

```

/NetSecurityWanted false def

```

Modifying the Root Menu

OpenWindows allows you to change the contents of your root menu. You need to create a SunView menu file that contains the menu entries you wish to appear in your OpenWindows root menu. You can then use the OpenWindows utility `buildmenu(1)` to translate the contents of the SunView menu file into NEWS toolkit code; this code builds the specified menu within OpenWindows.

The `buildmenu(1)` Utility

The `buildmenu(1)` utility builds an OpenWindows menu from a SunView menu file. The utility can be used either from a start-up file or from the shell prompt. See the reference manual page provided in Appendix A for more information.

Using the Default Menu File

OpenWindows provides a default menu file named `$OPENWINHOME/lib/openwin-menu`. Unless you specify otherwise, this file is used by the `init.ps` initialization file to create the OpenWindows root menu. Use the following call to `buildmenu`:

```
example% buildmenu -root -pinnable $OPENWINHOME/lib/openwin-menu | psh
```

Replacing the Default Menu File

If you wish to create your own menu file, proceed as follows:

1. Copy the file `$OPENWINHOME/lib/openwin-menu` to `~/openwin-menu`.
2. Make the appropriate modifications to `~/openwin-menu`. See the `buildmenu(1)` reference manual page for details.

When the server is initialized, `init.ps` searches for `~/openwin-menu` before it searches for `$OPENWINHOME/lib/openwin-menu`. If `~/openwin-menu` exists, it is used as the *menufile* argument to `buildmenu`:

```
example% buildmenu -root -pinnable ~/openwin-menu | psh
```

Note that if the user-created menu file `~/openwin-menu` does exist, the default menu file, `$OPENWINHOME/lib/openwin-menu`, is not used.

Modifying the Default Menu File

If you wish to use the default menu provided with the server, but wish to add a submenu of your own, proceed as follows:

1. Make sure that `~/openwin-menu` does not exist. (This ensures that the default menu file, `$OPENWINHOME/lib/openwin-menu`, is used.)
2. Create a SunView menu file containing the submenu information.
3. Insert in your `~/openwin-init` file a call to `buildmenu` that includes the submenu within the default menu. For example:

```
example% buildmenu ~/.mymenu -title "My Stuff" -pinnable | psh
```

Abbreviations

If you often type `N`ews commands directly to the server (see the section *Running News Programs* below) you may wish to define abbreviated versions of the commands you use most frequently. Note that you should never include these shortened names in client programs, since the definitions will only exist in the environment of your `.user.ps` file.

The following code can be added to `.user.ps` to redefine several common commands:

```

/ps {pstack} def                                % Alias redefinitions.
/cds {countdictstack =} def

/dbe {dbgbreakenter} def                        % Debugger redefinitions.
/dbx {dbgbreakexit} def
/dc {dbgcontinue} def
/dlb {dbglistbreaks} def
/dwb {dbgwherebreak} def

```

Changing the Root Background

If you are running in NeWS-only mode, or if you have defined `UsePaintRoot?` in your `UserProfile`, the `PaintRoot` procedure will be called to paint exposed areas of the root window. You can completely redefine the `PaintRoot` procedure in your `.startup.ps` file, but there is an easier way. The default `PaintRoot` procedure fills exposed areas of the root window with the color `RootColor`. You can change this color by putting lines such as the following in your `.user.ps` file:

```

/RootColor .5 dup dup rgbcolor store
# or
/RootColor ColorDict /MediumVioletRed get store

```

Note that the file `$OPENWINHOME/etc/NeWS/colors.ps` contains a list of all defined color names that you can use.

NOTE *The recommended way to change the root color is to use the OpenWindows Workspace Properties window.*

2.5. The UserProfile Dictionary

Some packages can be customized by the user. These packages look for a dictionary named `UserProfile` in `systemdict`. You can modify the contents of `UserProfile` in order to customize these packages. The packages typically inspect `UserProfile` only at server startup time; thus, you should modify `UserProfile` in your `.startup.ps` file. This ensures that your changes are present before packages look for them. It is not an error if values are missing from `UserProfile`. If a package looks for a value there and does not find it, the package uses a predefined default value.

Two examples of packages that use the `UserProfile` dictionary are *repeating keys* and *input focus*. The following entries are inspected:

Table 2-1 *UserProfile Entries*

<i>Key</i>	<i>Value</i>
FocusStyle	<i>/CursorFocus</i> for <i>focus-follows-mouse</i> or <i>/ClickFocus</i> for <i>click-to-type</i> .
KeyRepeatThresh	Length of time you have to hold down a key after which it starts repeating (in minutes).
KeyRepeatTime	Period with which a repeating key repeats (in minutes).
UsePaintRoot?	If true, you should disregard the X11 root background and call PaintRoot to paint exposed areas of the root window.

For example, putting the following code in your `.startup.ps` causes the server to wait 1/2 second before starting to repeat keys; it causes keys to repeat ten times per second (repeat period equals 1/10 of a second); it makes *focus-follows-mouse* the default focus style; and it causes **PaintRoot** to be called to paint the root background instead of using the X11 background:

```
UserProfile begin
  /KeyRepeatThresh 1 60 div 2 div def
  /KeyRepeatTime 1 60 div 10 div def
  /FocusStyle /CursorFocus def
  /UsePaintRoot? true def
end
```

NOTE *The recommended way to change the focus style and key repeat threshold is to use the OpenWindows Workspace Properties window.*

Another important use of the **UserProfile** dictionary is for customization of classes. Whenever a class is created, the class code looks in **UserProfile** for an entry whose key equals the name of the class. If the key exists and its value is a procedure, the procedure is run. The procedure can modify any of the variables or methods of the newly created class. The procedure should expect to find the class name and the class object on the operand stack and, after being executed, it should leave the class name and the presumably modified class object on the stack.

For example, suppose we have a class that knows how to draw stars. (Note that this example works only if the number of points in the star is odd.)

```

/Star [Object] [ ] classbegin

/LineWidth 5 def

/draw {                                     % n x y => -
  gsave
  LineWidth setlinewidth newpath moveto
  180 1 index div 180 sub
  exch 1 sub { 100 0 rlineto dup rotate } repeat
  pop closepath stroke
  grestore
} def

classend def

5 600 400 /draw Star send                % This draws the star.

```

This class draws a five-pointed star at location (600,400). The line width is set from a class variable, `LineWidth`. Suppose we want to customize this class so that its default line width is ten instead of five. We can add the following to the `.startup.ps` file:

```

UserProfile begin
  /Star {                                   % name class => name class
    dup /LineWidth 10 put
  } def
end

```

When the `Star` class is read in and built, the class package finds a procedure named `Star` in `UserProfile`. It calls this procedure with the class on the stack. The procedure modifies the name and class (in this case by changing a class variable) and returns the name and class on the stack.

2.6. Running NeWS Programs

Executing POSTSCRIPT Language Files

This section describes some of the ways in which you can run NeWS programs.

To execute a file that contains POSTSCRIPT language operators and NeWS extensions, use the `psh(1)` command (a manual page is provided for this command in `$OPENWINHOME/share/man` and in Appendix A).

The command establishes a connection to the X11/NeWS server and sends it the POSTSCRIPT files that you specify. (Note that this command can be used only if your program does not need to communicate with a C client side.) The command is demonstrated by the following example:

```

paper% cat > test.ps
/TestCanvas framebuffer newcanvas def
newpath
  0 0 moveto
  0 75 lineto
  75 75 lineto
  75 0 lineto
TestCanvas reshapecanvas
TestCanvas /Transparent false put
TestCanvas setcanvas
100 100 movecanvas
TestCanvas /Mapped true put
.2 fillcanvas
.1 sleep

paper% psh test.ps

```

The `psh` command causes the file `test.ps` to be executed: a canvas is mapped to the screen and removed after approximately a six-second pause.

Previewing POSTSCRIPT Graphics

You can preview a POSTSCRIPT file using the `pageview(1)` command. `pageview` renders the file, a page at a time, into an off-screen bitmap, which may be of arbitrary size, resolution, and orientation. You can adjust the size of the viewing window to see as much of the page as you want. You can also position the page within the window. See the `pageview` reference manual page in Appendix A for more information.

Communicating Directly with the Server

The News side of the X11/News server is a POSTSCRIPT interpreter with which you can communicate directly; this is particularly useful when you wish to debug existing code. To connect with the server, type `psh` to the shell prompt, then type `executive` in order to start an interactive session with the server. This is demonstrated by the following example:

```

paper% psh
executive
Welcome to X11/News Version 1.0

```

Once you have started an executive session, you can enter any POSTSCRIPT command:

```

340 1024 mul =
348160
/Celsius{
  32 sub 5 mul 9 div
} def
70 Celsius=
21.1111
32 Celsius=
0
currentcanvas =
canvas (width, height, root)
100 100 moveto
/Times-Italic findfont 24 scalefont setfont
(Hello world!) show                                % Prints string to root window.

10 setlinewidth
newpath
150 200 50 90 0 arc                                % Prints arc to root window.
stroke

```

Connecting to Remote NeWS Servers

NeWS is a network-based window system that allows you to connect to remote NeWS servers and display output on them. Note that according to NeWS terminology, the *server* runs on the machine with the display and keyboard, effectively providing the display and keyboard as resources for the *client* program. The environment variable `NEWSERVER` specifies the NeWS server to which local client programs connect. By default, they connect to the server on the local machine.

To display the output of a local client program on a remote machine, proceed as follows:

1. Request the owner/user of the remote machine to give your local machine permission to have a NeWS connection to the NeWS server on the remote machine. For example, if your local machine is “paper” and the remote machine is “neighbor”, the owner of “neighbor” can execute the following command:

```
neighbor% newshost add paper
```


2. Set the environment variable `NEWSSERVER` on your local machine to be the NeWS server on the remote machine. To perform this operation, you use the utility program `newsserverstr(1)`, which can be found in `$OPENWINHOME/bin` and returns the correct setting of `NEWSSERVER` for a given host. For example:

```
paper% setenv NEWSSERVER `newsserverstr neighbor`
```

When you now run a NeWS program on *paper*, the output is displayed on *neighbor*.

```
paper% psh client.ps
```

To display the output of local programs on your local machine again, you need to reset `NEWSSERVER`, or unset the variable using `unsetenv`.

```
paper% setenv NEWSSERVER `newsserverstr paper`
# or
paper% unsetenv NEWSSERVER
```

Running Remote Clients

It is more common to run a client program remotely and display its output locally than to run the client locally and display its output on a remote machine. The following list shows two ways to run a remote client. In each case, you must first make sure the remote machine permits a NeWS connection to the local machine. See the `newshost(1)` command above.

- Use `rlogin` to login to the remote machine, set the environment variable `NEWSSERVER` on the remote machine to be the server on the local machine, and run the client program on the remote machine. The client program can be an executable NeWS program or a file containing POSTSCRIPT operators and NeWS extensions, which you run using `psh(1)`. (See *Executing POSTSCRIPT Files*.)

```
paper% rlogin neighbor
.
.
neighbor% setenv NEWSSERVER `newsserverstr paper`
neighbor% newsclient_executable
# or
neighbor% psh newsclient.ps
```

- Use the `on(1)` remote execution service, which preserves environment variables like `NEWSSERVER`.

```
paper% setenv NEWSSERVER `newsserverstr paper`
paper% on neighbor newsclient_executable
```


2.7. Compiling Source Programs

If you are sharing a single copy of OpenWindows with other users, you should make a private copy of the sample source before trying to compile it. If your copy of OpenWindows is not writable by you, then you must make a private copy.

The following commands copy the sample source into `$HOME/junk` and compile it:

```
paper% cd $OPENWINHOME/share/src/xnews
paper% mkdir $HOME/junk
paper% tar cf - . | (cd $HOME/junk; tar xfbp - .)
```

2.8. Running a NeWS-Only Server

You can run the X11/NeWS server in NeWS-only mode, if the environment variable `NEWSONLY` is defined when you start the server. The NeWS-only server does not listen for X11 client connections, does not start up the X11 window manager, and does not offer X11-based applications for selection from the root and demo menus. X11 programs are unable to run in this environment. (See the *OpenWindows Installation and Start-Up Guide* for information about starting the server in NeWS-only mode.)

Using the X11 Protocol

Using the X11 Protocol	25
3.1. X11 Utilities	25
3.2. X11 Features Not Supported	26
3.3. Displaying POSTSCRIPT Images from X11 Clients	26
Using CPS	27
An Example Program	29
3.4. Connecting to Remote X11 Servers	29
3.5. Running Remote Clients	30
3.6. Restricted Access to the X11/NEWS Server	30
3.7. Running an X11-Only Server	30
3.8. X11 Idioms to Avoid	31
Black and White	31
GXset and GXclear	31
Pixmap Contents	31
Visuals	31
3.9. Compiling Source Programs	31



Using the X11 Protocol

The X11/NeWS server supports both the X11 and NeWS protocols; thus, it allows you to run applications written in either language. This chapter describes how the server supports X11 applications; it describes some of the standard X11 utilities, explains how to add X11 extensions to the server, and provides miscellaneous notes on using the X11 protocol. For more information on X11 programming, see the *Xlib Reference Manual*.⁵

3.1. X11 Utilities

The X11/NeWS distribution contains many of the standard utilities provided with the MIT X11 sample server. These utilities all cooperate with other X11 applications; however, not all of them cooperate with NeWS applications.

The following list contains some of the supported X11 utilities and indicates equivalent NeWS utilities that may also be used:

- `bitmap`: a bitmap editor. This produces code fragments for generating bitmaps in X11 programs. Note that you can use the XView program `iconedit` to create bitmaps for other XView programs.
- `xcalc`: a calculator.
- `xfd`: a font displayer.
- `xhost`: a server access control program. The NeWS shell script `newshost` gives the same control.
- `xlsfonts`: a font lister, which lists all fonts in the server.
- `xmodmap`: a keyboard modifier, which allows you to specify the mapping of keystations to keycodes. This can also be achieved by modifying POSTSCRIPT initialization files; for information, see the *NeWS Programmer's Guide* and Chapter 1 of this manual.
- `xpr`: a facility for printing a window dump.
- `xprop`: a property displayer. This utility is useful only with X11 windows, since NeWS windows do not have X11 properties associated with them.
- `xrdb`: an X resource database utility.

⁵ *Xlib Reference Manual*, O'Reilly & Associates, Inc., 1988

- `xrefresh`: a facility for refreshing the screen.
- `xset`: a facility for establishing user preferences. This can also be achieved by customizing POSTSCRIPT initialization files.
- `xsetroot`: the root window parameters utility.
- `xwd`: a facility for dumping a window-image. Note that the XView utility `snapshot(1)` can be used to dump screen regions; however, the X11 and Sun dump files are of different raster formats.
- `xwud`: an image displayer for X11. Note that the SunOS utility `screen-load(1)` can be used to load a Sun rasterfile onto the screen. The NeWS utility `showimage(1)` can also be used to display a Sun rasterfile. As indicated above, X11 and Sun rasterfiles are of different formats.
- `xwininfo`: a window information facility. Note that this does not display information about NeWS windows and canvases; if you attempt to use it in this way, it displays information on the root canvas only.

3.2. X11 Features Not Supported

The following applications and libraries, all of which are available from MIT, run on the X11/NeWS server but are not supported by Sun Microsystems, Inc.

- Uncommon Xlib-based applications
- The X toolkit and X toolkit applications (such as `xbiff`, `xclock`, `xedit`, `xload`, `xlogo`, and `xmh`)
- Andrew
- The `uwm` and `wm` window managers
- The CLX Common Lisp interface

3.3. Displaying POSTSCRIPT Images from X11 Clients

The X11/NeWS server allows X11 programs to perform POSTSCRIPT imaging inside X11 windows. The current implementation of this facility is preliminary and will be replaced by a standard interface when one is adopted by the X Consortium.

To display POSTSCRIPT images, an X11 client must have not only the normal X11 connection, but also a NeWS connection through the CPS interface. (See the *NeWS Programmer's Guide*.) POSTSCRIPT language code is sent through the NeWS connection, and X11 requests are sent through the X11 connection.

This method works for most graphics operations. However, the following points should be noted:

- Since POSTSCRIPT and X11 requests are being sent through different communication channels, synchronization can be a problem. All NeWS calls must be flushed with `psio_flush_PostScript()` to ensure that POSTSCRIPT graphics become visible. (See the *NeWS Programmer's Guide*.) Note that X11 graphics do not usually need to be flushed, since the programs in which they are defined normally use some form of event looping that flushes the graphics requests while waiting for the next event. However, if the results of executing some POSTSCRIPT language code depend on completion of some X11 requests, a call to `XSync()` is required to flush the X

buffers and confirm that the operation has completed.

- Since X11 uses an upper-left origin, and NeWS uses a lower-left origin by default, the default transforms cause graphic images to be displayed upside-down. Therefore, when drawing an X11 canvas, you can solve this problem by resetting the current transformation. This is demonstrated by the following example:

```

clippath pathbbox 0 exch translate pop pop pop
                                     % Move the origin to upper
                                     % left.

1 -1 scale                            % Reverse coordinate
                                     % system so that increasing
                                     % Y is down.
```

- Some NeWS operations are inappropriate for X11 windows (for example, operations that make a window round) and can have unpredictable results, as X11 may not comprehend them.
- Windows are the only resources NeWS and X11 should need to share. Attempts to share other resources may have unforeseen consequences and are likely to fail.
- The NeWS `canvastype` dictionary automatically assumes several keys when the X11 components of the server are present; however, if these components are not present, the extra keys disappear. Therefore, before attempting to use the keys, ensure that the X11 interpreter is present.
- Do not attempt to use NeWS to send input to an X11 client.

Using CPS

Before attempting to use CPS to specify NeWS graphics in your X11 program, note the following requirements:

- The following macros enable the client to send resource IDs down the NeWS connection.

```

/* client is in top twelve bits,
 * 13th is reserved for the server,
 * object uses lower 19 bits
 */
#define client_id(x)  ((int)((x) & 0xfff00000) >> 20)
#define object_id(x) ((x) & 0x0007fff)
```

As shown, the client ID is stored in the top twelve bits in any X11 resource ID; the object ID is the lower 19 bits (this is true for any X11 object that is a resource).

- The NeWS code that you specify needs to know which X11 window you wish to draw in. Thus, it must access the X11 resource database for the X11 client application and retrieve IDs from it. Access to the X11 resource database is

in the form of the NeWS class **XResource**. (Information on the NeWS class system is provided in the *NeWS Programmer's Guide*.)

The following **XResource** methods can be used to notify NeWS code of an X11 window:

/new

– **/new dict**

Returns a new instance of the X resource database.

Example

```
% get a new instance of X11 resource DB
/initresDB { % - => -
    /resDB /new XResource send def
} def
```

/XLookupClient

XclientID **/XLookupClient** –

Generally set to class **XResource** itself; gets the given X client's resource database, in which you can look up any of its windows. It takes an X client ID, which can be pulled out of an X Window ID by using the `client_id(Xwin)` macro above.

Example

```
% get this client's resource DB
/initclientDB { % XclientID => -
    /clientDB exch /XLookupClient XResource send def
} def
```

/XConstructID

XobjectID **/XConstructID** realID

Returns a *realID* for the given *XobjectID*. The *realID* here is a combination of the *clientID* and the *objectID*. It takes an X Window ID, which is generated using the `object_id(Xwin)` macro above. Due to a limitation in the way NeWS represents integers, the upper three bits are lost in transmission. That is, only 16 of the 19 *objectID* bits may be used by NeWS.

/XLookupID

XobjectID **/XLookupID** any

Given an X11 realID, this returns the NeWS object for it.

Example Using this together with `/XConstructID` you can define an operator to return the NeWS object for a given X11 object ID:

```
% finds given id and leaves on stack
/getobj { %XObjectID => obj
        /XConstructID clientDB send           % realID
        /XLookupID clientDB send           % object
} def
```

An Example Program

The ‘Logos’ program in `$OPENWINHOME/demo` draws using X11 calls and the POSTSCRIPT language in the same window. Its source is in `$OPENWINHOME/share/src/xnews/client/X11/logo{.c,.cps}`.

3.4. Connecting to Remote X11 Servers

This section describes how to run a client program locally and display its output on a remote machine. The following section describes the more common operation of running a client program remotely and displaying its output on your local machine.

The environment variable `DISPLAY` specifies the X11 server to which client programs connect. By default, they connect to the server on the same machine they run on. To display the output of a local client program on a remote machine, proceed as follows:

1. Request the owner/user of the remote machine to use the `xhost(1)` command to give your local machine permission to have an X11 connection to the X11/NeWS server on the remote machine. For example, if your local machine is “paper” and the remote machine is “neighbor”, the owner should execute the following command:

```
neighbor% xhost + paper
```

2. Set the environment variable `DISPLAY` on your local machine to be the X11 server on the remote machine. For example:

```
paper% setenv DISPLAY neighbor:0
```

When you now run an X11 program on *paper*, the output is displayed on *neighbor*.

To display the output of local programs on your local machine again, you need to reset `DISPLAY`.

```
paper% setenv DISPLAY paper:0
```

3.5. Running Remote Clients

The following list shows three ways to run a remote client. In each case, you must first make sure the remote machine permits an X11 connection to the local machine. See the `xhost(1)` command above.

- Use `rlogin` to login to the remote machine, set the environment variable `DISPLAY` on the remote machine to be the server on the local machine, and run the client program on the remote machine.

```
paper% rlogin neighbor
.
.
neighbor% setenv DISPLAY paper:0
neighbor% x_client
```

- Use `rsh(1)` to perform the same task.

```
paper% rsh neighbor x_client -display paper:0
```

- Use the `on(1)` remote execution service, which preserves environment variables like `DISPLAY`.

```
paper% setenv DISPLAY paper:0
paper% on neighbor x_client
```

3.6. Restricted Access to the X11/NEWS Server

The X11 protocol supports an access control list of all hosts permitted to establish connections with the server. This list is manipulated by the `XAddHost()`, `XAddHosts()`, `XRemoveHost()`, and `XRemoveHosts()` calls in `Xlib`. Access checking is turned on and off by the `XDisableAccessControl()` and `XEnableAccessControl()` calls. The `xhost(1)` program is an X11 client that changes the access control list.

In the X11/NEWS server, the X11 routines and `xhost` use the same `RemoteHostRegistry` dictionary. If a host is allowed access to the server from X11, it is also permitted to access the server using NEWS protocol. See the *Modifying the Access List* section in Chapter 2, for more information on access lists.

3.7. Running an X11-Only Server

The X11/NEWS server can be run in X11-only mode. If the environment variable `X11ONLY` is defined when you start the server, the server initializes itself as an X11-only server. This means that the server does not listen for NEWS connections and does not start the X11/NEWS window manager. (See the *OpenWindows Installation and Start-Up Guide* for information on starting the server in X11-only mode.) NEWS programs are unable to run in this environment.

If you run the server in X11-only mode, you must start your own window manager. The server configuration file `.openwin-init` continues to be sourced in X11-only mode, and is the suggested location from which to start your X11 window manager. To allow the window manager time to initialize itself, you should put a ten-second sleep in `.openwin-init` immediately after starting

the window manager and before starting any client program.

If you set the `USE_MIT_VISUALS` environment variable, the X11/NeWS server exposes a PseudoColor visual and colormap as the default on color displays (as does the MIT X11 sample server). As a result, many X11 programs using non-portable color will work, but the NeWS color model will break.

3.8. X11 Idioms to Avoid

The implementation of X11 in X11/NeWS conforms to the X11 protocol.⁶ Note, however, that the specification leaves some leeway in implementation, and not all servers offer the same design decisions. Specifically, programs that assume the MIT sample implementation may break. This section indicates some of the problems that may be encountered.

Black and White

In X11/NeWS, the colors black and white are not defined as 0 and 1 (or vice versa). To specify black and white, clients should always use the X11 macros `BlackPixel()` and `WhitePixel()`.

GXset and GXclear

`GXset` and `GXclear` are two of the possible RasterOp function codes that you can set in X11 graphics contexts. `GXclear` sets the destination pixels to 0, `GXset` sets all bits of the destination pixels to 1. In X11/NeWS, `GXclear` clears to whatever the color is in the first slot of the colormap; `GXset` sets to whatever the color is in the last slot. Consequently, these two RasterOp modes must be used with care.

Pixmap Contents

The contents of an X11 pixmap are undefined when it is first created. X11/NeWS does not clear it to black or white at creation; thus, the pixmap contains whatever random pattern was in the memory that was allocated to it by X11/NeWS.

Visuals

The X11/NeWS server uses a different collection of visuals than the MIT sample server. See the *Color Support* chapter of this guide for more information.

3.9. Compiling Source Programs

If you are sharing a single copy of OpenWindows with other users, you should make a private copy of the sample source before trying to compile it. If your copy of OpenWindows is not writable by you, then you must make a private copy.

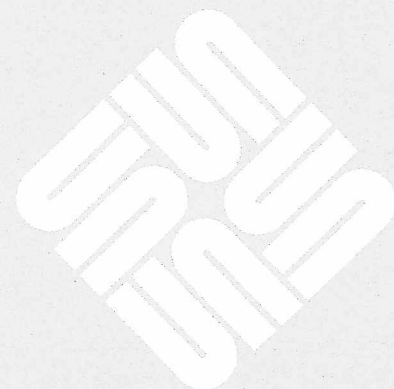
The following commands copy the sample source into `$HOME/junk` and compile it:

```
paper% cd $OPENWINHOME/share/src/xnews
paper% mkdir $HOME/junk
paper% tar cf - . | (cd $HOME/junk; tar xfbp - .)
```

⁶ The X11 protocol is defined in *X Window System Protocol, Release 3* by Robert W. Scheifler.

Window Management

Window Management	35
4.1. External Window Management	35
4.2. NeWS Window Management	35
4.3. X11 Window Management	36
Using Redirection	37
Example	37
Example	37
The <code>override-redirect</code> Attribute	38
4.4. Window Management in X11/NeWS	38
4.5. Switching Window Managers	38
X11 Window Managers and NeWS Windows	39
Restarting <i>pswm</i>	39
Starting With Other Window Managers	39
4.6. Problems with External Window Management	40



Window Management

Window management is a set of functions with which a user can control the appearance of windows on the screen. The agent that implements these functions is called the *window manager*. The functions include moving, resizing, opening, closing, raising, lowering, and quitting windows. Other functions of window managers include setting input focus, installing color maps, and starting up new applications.

A window manager typically provides these functions by placing *decorations* around the application's windows. Typical decorations are a window's header (or *namestripe*) and its resize and close symbols; most window managers provide *pop-up menus* for additional functions. Usually, the window manager also provides a pop-up menu that is available over the root window. This menu allows the user to display other windows and applications.

4.1. External Window Management

The X11 and NeWS windowing environments support the concept of *external window management*. In this concept, certain window management functions are implemented separately from any application. This contrasts with SunWindows, for example, where window management functions are built into every application. In NeWS, the window management functions reside in the NeWS toolkit's class hierarchy. In X11, they are implemented by a special client named the *window manager*. This client manages the windows that other X11 clients create.

Since the X11 window manager is merely a client, it can be replaced by a different window manager. This can even be done dynamically, by killing the current window manager and starting another.

4.2. NeWS Window Management

To create a window in NeWS, you must create an instance of **ClassCanvas** and send this instance to **ClassFrame** as a parameter of the `/newdefault` message. This creates a new *frame object*, which wraps itself around the client canvas and starts managing it. The frame and client canvases together form the window. (See the *NeWS Programmer's Guide* for a discussion of classes in NeWS.)

The following programming example creates an instance of **ClassBaseFrame**, which manages an instance of a subclass of **ClassCanvas** named **MyCanvas**. The behavior of **MyCanvas** is different from that of **ClassCanvas** in several ways. The colors used for painting the background and border are set to yellow and red respectively, and the border width is set to 10. These changes become apparent when the `/PaintCanvas` method is called in order to repaint the canvas

inside the frame when it is damaged.

```

%
% Create a subclass of ClassCanvas to be customized.
%

/MyCanvas ClassCanvas
dictbegin % Instance variables
  /FillColor 1 1 0 rgbcolor def      % Make fill color yellow.
  /StrokeColor 1 0 0 rgbcolor def    % Make stroke color red.
  /BorderStroke 10 def               % Make border width = 10.
dictend
classbegin % Class variables and methods
  % This is the default PaintCanvas...
  /PaintCanvas { % - => -
    StrokeColor BorderStroke FillColor
    /StrokeAndFillCanvas self send
  } def
  /preferredsize { % - => -
    /preferredsize super send
    100 max exch
    100 max exch
  } def
classend

%
% Create a new instance of ClassBaseFrame, which will manage
% an instance of the class defined above, MyCanvas.
%

/win [MyCanvas] [] framebuffer /newdefault ClassBaseFrame send def

(Label) /setlabel win send          % Set the frame label.
/place win send                     % Put up window with preferred size.
/map win send                       % Make visible.
/activate win send                  % Start event handlers.
/win null def                       % Get rid of our frame reference.

```

All the window management functions that NEWS provides are implemented by the *frame* object. Whenever you move or resize a window, the frame is the part of the window that actually responds to your mouse-movements. The frame then sends appropriate messages to the client canvas and modifies it accordingly.

4.3. X11 Window Management

Window management in X11 is based on allowing one client (that is, the window manager) to intercept certain operations that other clients are performing. The operations that are intercepted (or *redirected*, to use X11 terminology) are those that the window manager must evaluate and modify, in order to maintain control over the current windows.

It is possible for X11 window managers to work without using the redirection facilities: this type of window manager *grabs* certain combinations of modifier

keys and mouse buttons, and uses them to manipulate windows. However, not using redirection severely limits the window manager's user interface; in particular, it is impossible for such a window manager to add decorations around the outside of an application's window. Since this kind of window manager is unlikely to be useful, the remainder of this chapter only discusses those window managers that use redirection.

Using Redirection

To use redirection, the window manager selects for **SubstructureRedirect** on the window in which it will manage application windows. Typically, this window will be the root window, since applications will normally create windows as children of this window. **SubstructureRedirect** allows the window manager to intercept X11 requests that would change the appearance of windows on the screen. The two most important such requests are the **MapWindow** and **ConfigureWindow** X11 protocol requests. Only one client can select for **SubstructureRedirect** at a time; thus, only one window manager can run at a time. Windows that are created as grandchildren of the root window are not affected by redirection; thus, an application can create interior windows without inhibition.

An application is free to create windows as it wishes. However, when it tries to map the window (using the **MapWindow** request) or change its location or size (using **ConfigureWindow**), the operation is not performed. Instead, the server generates a **MapRequest** or **ConfigureRequest** event and sends it to the window manager. The window manager can examine the state of the screen and make arbitrary changes to the state of this or any other window on the screen. In response to one of these events, the window manager is free to carry out the request, modify the request, or do nothing. **MapWindow** and **ConfigureWindow** requests sent by the window manager are not redirected back to the window manager; they are executed by the server.

Example

Suppose an application attempts to map a window using the **MapWindow** request. The server generates a **MapRequest** event and sends it to the window manager. The window manager may wish to provide decorations around this window, and therefore it does not map it immediately. Instead, it creates another window (a *decorator window*) and places decorations in it. The window manager then reparents the application's window into the decorator window and maps them both. This method allows all applications to have their windows decorated and managed by one window manager.

Example

Suppose there is a window manager that enforces a policy of not allowing windows to overlap. Suppose further that an application has just issued a **ConfigureWindow** request to resize one of its windows. If this request were performed, it might cause the window to obscure another window. This window manager would use redirection to prevent this from happening. When it receives the **ConfigureRequest** event that results from the application's **ConfigureWindow** request, the window manager might choose to move or resize other windows so that the application's window does not obscure them. Or, the window manager might choose to refuse this resize request, or alter it so that no overlapping occurs.

The override-redirect Attribute

It is possible for applications to create windows as children of the root window and have them not be managed by the window manager. The application does this by setting the `override-redirect` attribute on the window. This capability is useful for pop-up menus, which typically are not decorated in the same way as more permanent windows.

For more information about how applications should behave in an externally window-managed environment, see the *X11 Inter-Client Communication Conventions Manual (ICCCM)*.⁷ This manual covers, among other things, how X11 applications can give “hints” about what kind of services they would like from the window manager.

4.4. Window Management in X11/News

The X11/News window manager, *pswm*, manages X11 windows using the News toolkit Frame objects, as described in section 4.2. *pswm* relies on the frame and canvas classes of the News toolkit to do its work, and it inherits the user interfaces and graphics capabilities they provide. This allows X11 and News windows to be managed with the same user interface, providing a seamlessly integrated environment.

From an X11 application’s point of view, *pswm* looks just like any other X11 window manager. *pswm* has selected for `SubstructureRedirect` on the root window, and it acts on redirected requests in the typical X11 fashion, as described above. However, the implementation of *pswm* is completely different. Part of *pswm* is implemented as News lightweight processes inside the X11/News server, and the other part is implemented as an ordinary UNIX process.

4.5. Switching Window Managers

It is possible to switch window managers simply by stopping the first window manager and starting the second one. To stop a window manager, you must find its UNIX process-id, and then send it a signal using the UNIX `kill` command. You can do this using the `ps` and `grep` utilities. For example, suppose you want to find the id of the currently running *pswm* process:

```
example% ps ax | grep pswm
7840  p5  S   0:00  grep pswm
7710  p6  IW  0:00  pswm -init
```

When you have found the process id of *pswm*, you can shut it down by sending it a `TERM` signal.

```
example% kill -TERM 7710
```

You can now run another window manager. If you follow this procedure, there will be a short period of time where there is no window manager running. This is a potentially dangerous situation, because without a window manager, there may be no way to move the input focus, and there may be no way to start up new applications. If shutting down the window manager has left you without a shell

⁷ *X11 Inter-Client Communication Conventions Manual (ICCCM)*, Sun Microsystems, Inc., May 1989.

to type commands, you are stuck.

There are several ways of avoiding this. One approach is to kill the old window manager and start up the new one on a single command line. Suppose you want to stop *pswm* and then run a different window manager named *xwm*. The following command is reasonably safe:

```
example% kill -TERM 7710; xwm &
```

This kills the currently running invocation of *pswm*, and then immediately runs *xwm*. This avoids the situation where there is no place to type and no window manager running.

X11 Window Managers and NeWS Windows

If you run another X11 window manager, it manages all X11 applications on that server. NeWS applications running currently or in the future are not managed by the X11 window manager; they continue to be managed by the Frame objects of the NeWS toolkit.

NOTE *NeWS applications manage the input focus in much the same way as they manage windows. If an X11 window manager is running at the same time as NeWS applications, they may conflict over the input focus. These situations are not fatal, but they are inconvenient. A typical problem might be a NeWS application unexpectedly grabbing the input focus away from an X11 application.*

Restarting *pswm*

If you have been running other window managers, and you wish to return to using *pswm*, you can simply use the shell to restart *pswm* in the background:

```
example% pswm &
```

For *pswm* to work, both the `DISPLAY` and the `NEWSERVER` environment variables must be set to point to the same server. This is necessary because *pswm* makes both X11 and NeWS connections to the X11/NeWS server. If, for example, the `NEWSERVER` environment variable indicated the X11/NeWS server on the local host, but the `DISPLAY` environment variable indicated an X11 server on a remote host, *pswm* would make connections to two different servers. If this occurs, *pswm* will print an error message and exit. Since *pswm* requires an X11 and NeWS connection to the same server, it is not possible to run *pswm* on any X11 server other than X11/NeWS.

Starting With Other Window Managers

The X11/NeWS server starts up *pswm* automatically. If you wish to run only X11 applications, and you wish also to run another window manager, you can start the server in X11-only mode. (See *Running an X11-Only Server* in Chapter 3.) To start a different window manager in X11-only mode, simply place a suitable command line in your `.openwin-init` file. (See the *OpenWindows Installation and Start-Up Guide*.)

4.6. Problems with External Window Management

There are a number of problems that can occur in an environment with external window management. Programmers must be careful to avoid these problems.

- Grabbing the server

If your application grabs the server, it cannot manipulate windows unless they have `override-redirect` set. Since it has grabbed the server, it prevents the window manager from processing requests. A program can deadlock the system if it grabs the server and then executes a request that is redirected to the window manager.

- Mapping windows

You cannot assume that the window is mapped immediately after you issue a `MapWindow` request. For example, if you map a window and then set the focus to it, the mapping may fail. The reason is that the `MapWindow` request is redirected to the window manager instead of mapping the window immediately. The window manager takes a certain amount of time to respond and its response is asynchronous. Requests that immediately follow the `MapWindow` request are executed before the window manager actually maps the window. You must wait until you receive a `MapNotify` event or an `Exposure` event before you can issue any requests that assume the window is mapped.

- Requesting size and position of windows

Do not assume that you will get the window size or position that you request. The layout policy implemented by the window manager may not permit this. If you do something that depends on the size of the window, you should wait until you get a `ConfigureNotify` event, and use the information in it. This feature is meant to empower users by giving them a choice of window managers; it is not meant to be restrictive.

- Changing window managers

You can kill and start window managers at will. If you kill a window manager while you have no means to start up another one, you may be left in a state where you cannot do anything. To avoid this state, use the technique described in the previous section. Alternatively, you can stop and start the window managers from a `News psterm` window or from a `SunView shelltool`. As a last resort, you can login from another machine and start a window manager.

- Conforming to the ICCCM

You should write your application to work as well as possible with any window manager that conforms to the ICCCM. Applications or window managers that violate the ICCCM should be fixed. This will help ensure that they are inter-operable with other ICCCM-conforming window managers and applications.

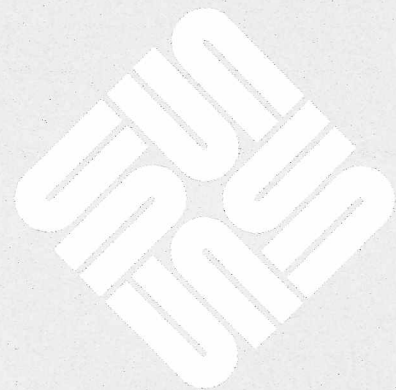
- Depending on a particular window manager

Your application should not rely on the behavior of a particular window manager. Different window managers may have differing behavior, even

though both conform to the ICCCM. You should depend only on behavior that is guaranteed to be present in ICCCM-conforming window managers.

Font Support

Font Support	45
5.1. Server-Supplied Fonts	45
Locating Fonts	45
The Core Set of Outline Fonts	46
Times	46
Helvetica	46
Helvetica Narrow	46
Symbol	46
Courier	46
New Century Schoolbook	47
Palatino	47
ITC Bookman	47
ITC Avant Garde	47
ITC Zapf Dingbats	48
ITC Zapf Chancery	48
Lucida Family	48
Bembo	48
Rockwell	48
Gill Sans	48
The Minimum Set of Fonts for the Server	49
5.2. Font Families	49
5.3. Accessing and Scaling Fonts	49
5.4. Using Other Fonts	50



5.5. Adding Fonts	51
Adding F3 Fonts	51
Generating Bitmap Files	51
5.6. Converting NeWS 1.1 Fonts for Use with the X11/NeWS Server	52
5.7. Font Aliases	52
5.8. Fonts in the X11 Window System	52
Using OpenFonts with the X11 Protocol	52
X11 Access to NeWS Fonts	53
5.9. Font Limitations	53

Font Support

The X11/NeWS server supports a large variety of fonts, such as standard English text fonts, symbol fonts, and foreign language fonts. A set of fonts is provided with the server; other fonts can be added. In addition, the server supports two types of font representation. This chapter describes the representations supported, the system-supplied fonts, and the use of font conversion tools.

5.1. Server-Supplied Fonts

The fonts provided by the X11/NeWS server are contained in the directory `$OPENWINHOME/lib/fonts`. There are two types of font representation: *outline fonts* (named **OpenFonts™**) and *bitmap fonts*. Both types of font representation can be used and shared by X11 and NeWS.

- Outline fonts

Each outline font is provided as an OpenFonts F3™ format scalable outline file. This means a single outline font can be arbitrarily scaled to a variety of sizes, thus replacing a whole family of bitmap files. The typical filename extension for these files is `.f3b`. These fonts can be used to generate arbitrarily transformed and rotated characters from NeWS. X11 programs use these fonts to get characters that are arbitrarily transformed, but are not rotated. When you use a character from an F3 font for the first time, the bitmap must be created; then it can be cached and reused. If initial display-time for previously undisplayed characters is critical, a bitmap font may be preferable.

- Bitmap fonts

Each size-specific bitmap font is provided as a file in binary format; this format is native to the server. The typical filename extension for these files is `.fb`. For smaller sizes, some fonts provide a hand-tuned bitmap. The tuned bitmap provides superior results and should be used, if available.

NOTE Since the bitmap font files are binary files, they cannot be shared between machines of different architectures (for example, Sun4 and Sun 386i).

Locating Fonts

The X11/NeWS server uses the value of the environment variable `FONTPATH` to locate fonts. The fonts provided with the server are located in the default font directory, `$OPENWINHOME/lib/fonts`, which is the default value of `FONTPATH`. You may add fonts to the default font directory, and you may create additional font directories.

FONTPATH is a colon-separated list of directories, which the server searches in order when looking for a particular font. You can modify FONTPATH from X11 by using XSetFontPath () or the shell command xset:

```
example% xset -fp newfontpath
```

From NeWS, use the BuildFontPath operator to modify FONTPATH:

```
(path1:path2:path3) BuildFontPath
```

The Core Set of Outline Fonts

The X11/News server provides a core set of 57 F3 format typefaces. These fonts contain outline and typographic information that preserves the essential properties of a typeface when characters are scaled or rotated. The core set includes the 35 fonts commonly found in the Apple LaserWriter[®] Plus printer.⁸ They are compatible in appearance, metrics, character set, and trademarks with the LaserWriter fonts.

The descriptions of the F3 format fonts below should assist you in choosing the proper font for your needs.

Times	Times [™] , the most widely used typeface in the world, was originally designed for use in newspapers. ⁹ It is a good choice for manuals, books, or any application involving long sections of text.
Helvetica	Helvetica [™] is a clean, modern design that is best for headings, displays, or short passages of text. ¹⁰
Helvetica Narrow	This is a compressed design that is used when space is at a premium, such as in captions to illustrations. It is hard to read except for short texts.
Symbol	Symbol contains characters for the setting of mathematics. The character set for the Symbol font is listed in the file Symbol.map.
Courier	Courier is a digital adaptation of a typewriter face and is a constant-width, or monospaced, font. It allows for easier alignment of text in applications such as computer programs, or when the appearance of a letter made by a typewriter is required.

⁸ LaserWriter Plus is a registered trademark of the Apple Computer Corporation.

⁹ Times is a trademark of the Linotype AG and/or its subsidiaries.

¹⁰ Helvetica is a trademark of the Linotype AG and/or its subsidiaries.

New Century Schoolbook

Long favored by publishers for use in school textbooks because of its high degree of legibility, New Century Schoolbook is a good choice for books, manuals, and newsletters. Because of its design, it is also a good choice for use on computer screens.

Figure 5-1 The Core Set of Outline Fonts

<p>AvantGarde-Book <i>AvantGarde-BookOblique</i> AvantGarde-Demi <i>AvantGarde-DemiOblique</i> Bembo Bembo-Bold <i>Bembo-BoldItalic</i> <i>Bembo-Italic</i> Bookman-Demi <i>Bookman-DemiItalic</i> Bookman-Light <i>Bookman-LightItalic</i> Courier Courier-Bold <i>Courier-BoldOblique</i> <i>Courier-Oblique</i> GillSans GillSans-Bold <i>GillSans-BoldItalic</i> <i>GillSans-Italic</i> Helvetica Helvetica-Bold <i>Helvetica-BoldOblique</i> Helvetica-Narrow Helvetica-Narrow-Bold <i>Helvetica-Narrow-BoldOblique</i> <i>Helvetica-Narrow-Oblique</i> <i>Helvetica-Oblique</i> Lucida-Bright</p>	<p>Lucida-BrightDemiBold <i>Lucida-BrightDemiBoldItalic</i> <i>Lucida-BrightItalic</i> LucidaSans LucidaSans-Bold <i>LucidaSans-BoldItalic</i> <i>LucidaSans-Italic</i> LucidaSansTypewriter LucidaSansTypewriter-Bold <i>NewCenturySchlbk-BoldItalic</i> <i>NewCenturySchlbk-Italic</i> NewCenturySchlbk-Roman Palatino-Bold <i>Palatino-BoldItalic</i> <i>Palatino-Italic</i> Palatino-Roman Rockwell Rockwell-Bold <i>Rockwell-BoldItalic</i> <i>Rockwell-Italic</i> Σμβολ (Symbol) Times-Bold <i>Times-BoldItalic</i> <i>Times-Italic</i> Times-Roman ZapfChancery-MediumItalic *⊠⊡⊣⊥⊦⊧⊨⊩⊪⊫⊬⊭⊮⊯⊰⊱⊲⊳⊴⊵⊶⊷⊸⊹⊺⊻⊼⊽⊾⊿ (Zapf Dingbats)</p>
--	---

Palatino

One of the most popular of all the fonts available for laser printing, Palatino™ is based on an elegant design that is especially good in formal documents, such as books, presentations, or corporate communications.¹¹ The italic has a *calligraphic* feel and works well on invitations and posters.

ITC Bookman

ITC Bookman™ is good for use in less formal applications such as flyers and advertising.¹² It also works well on low-resolution computer screens.

ITC Avant Garde

ITC Avant Garde™ is another face that works well when a distinctive design but less formality and readability is needed.¹³ It is commonly used in advertising for large display of short amounts of text.

¹¹ Palatino is a trademark of the Linotype AG and/or its subsidiaries.
¹² ITC Bookman is a trademark of International Typeface Corporation.
¹³ ITC Avant Garde is a trademark of International Typeface Corporation.

ITC Zapf Dingbats	ITC Zapf Dingbats™ are symbols such as bullets, pointing hands, stars, and decorations. ¹⁴ They help to draw attention to certain parts of documents where special emphasis is required. The Dingbats character set is listed in <code>Dingbats.map</code> .
ITC Zapf Chancery	ITC Zapf Chancery™ is a font that imitates handwritten calligraphy. ¹⁵ It is good for formal invitations, menus, or large, informal display of short amounts of text.
Lucida Family	The core set also includes the Lucida® typefonts, which are used in the OPEN LOOK graphical interface. ¹⁶ These faces were designed specifically for good appearance on low-resolution digital devices. They are a coordinated family of designs that includes a sans serif, a serif, and a typewriter font (Lucida Sans, Lucida Bright, and Lucida Sans Typewriter, respectively).
Bembo	Bembo™ has long been a favorite for high-quality publications, such as books and magazines. ¹⁷ The design is an adaptation of a classical Renaissance typeface.
Rockwell	Rockwell™ is a square serif font and a good choice for headings. ¹⁸ In large display, Rockwell has a strong, emphatic design.
Gill Sans	Originally designed (and still used) for the signs in the London Underground, Gill Sans™ is an excellent choice when a sans serif is needed for text or display. ¹⁹ Gill Sans is more distinctive than Helvetica and is therefore a favorite of graphic designers for business cards, product packages, and signs.

¹⁴ ITC Zapf Dingbats is a trademark of International Typeface Corporation.

¹⁵ ITC Zapf Chancery is a trademark of International Typeface Corporation.

¹⁶ Lucida is a registered trademark of Bigelow and Holmes.

¹⁷ Bembo is a trademark of Monotype Ltd.

¹⁸ Rockwell is a trademark of Monotype Ltd.

¹⁹ Gill Sans is a trademark of Monotype Ltd.

The Minimum Set of Fonts for the Server

Due to space constraints, users may elect to install a subset of the core set of fonts in the server. These fonts are the common fonts for POSTSCRIPT previewing, XView programs, and X11 applications.

Table 5-1 *Minimum Set of Fonts*

<i>Outline fonts</i>	<i>Bitmap fonts</i>
Courier	fixed
Courier-Bold	cursor
Courier-BoldOblique	olglyph
Courier-Oblique	Icon
Helvetica	NeWSCursor
Helvetica-Bold	OLCursor
Helvetica-BoldOblique	Screen
Helvetica-Oblique	Screen-Bold
LucidaSans	
LucidaSans-Bold	
LucidaSans-BoldItalic	
LucidaSans-Italic	
LucidaSansTypeWriter	
LucidaSansTypeWriter-Bold	
Times-Bold	
Times-BoldItalic	
Times-Italic	
Times-Roman	

5.2. Font Families

A font family can consist of any group of related fonts. For example, a family might include an F3 format font and a number of bitmap files to be used when rendering the font at particular sizes. In the case of bitmap fonts, a family might be a set of files that provide the same typeface in different sizes.

The server needs to know which groups of files constitute a font family. The file `Families.list` contains a master list of font families. Each font family is represented by a file whose name consists of the name of the font and the extension `.ff`.

The font family files and the master list of families are generated by the `bldfamily(1)` program. (See the `bldfamily(1)` manual page in Appendix A.) Once created, each file is a key to the fonts in a family and is associated with a POSTSCRIPT language name (such as `Helvetica-Oblique`).

5.3. Accessing and Scaling Fonts

When a particular font name is specified in a program, for example `Helvetica-Oblique`, the server locates the font family with the same name. Information in the family is also used to scale a font to obtain a particular point size.

□ Bitmap fonts

The system-supplied bitmap fonts include several versions of each named font; each version is a different point size. When a specific font name and

point size are specified, the associated family is searched for the particular point size. If no font is found, the server uses the font whose size is closest to the specification without being larger.

It is possible to scale a bitmap font, using commands such as the POSTSCRIPT language operator `scafont`. The server finds the associated family, and the font with the closest size is used to generate a new font of the requested point size. The new font is added to the family.

Thus, for bitmap fonts, requesting a font of a particular point size may result in a smaller point size being used; requesting that a font be scaled results in a font of the requested size being generated (if it does not already exist).

- Outline fonts

The POSTSCRIPT language is scale- and resolution-independent in all respects, including that of font size specification. Thus, for any outline font, the specified name is used to find the associated font family; it is then scaled to the size specified. For outline fonts, a request for a particular point size and a request that a font be scaled to achieve a particular point size invoke the same processing. The result is always a font in the size requested.

5.4. Using Other Fonts

To use additional fonts with the X11/NeWS server they must be converted to an appropriate format with the `convertfont(1)` program. This program reads in a set of named font files and dumps them out in a user-specified format. The program can read font files that have the following formats:

- Adobe ASCII bitmap format
- Adobe ASCII metric format
- NeWS 1.1 binary format
- X11/NeWS binary format
- X11 BDF bitmap format
- `vfont(5)` binary format

The program can convert the font files to the following formats:

- Adobe ASCII bitmap format
- NeWS 1.1 binary format
- X11/NeWS binary format (the default)
- `vfont(5)` binary format
- X11 BDF bitmap format

The preferred font format for exchanging bitmap files is Adobe bitmap font format. By default, the server uses version 2.1 of this; it also recognizes both the X11 interpretation of this version and the earlier 1.0 and 1.4 versions. See the manual page `convertfont(1)` for further information.

NOTE The font directory may contain binary fonts that use an older format, favored by previous versions of NeWS. The `convertfont` program handles these fonts.

5.5. Adding Fonts

You may add fonts to augment the core set of fonts that comes with the X11/NeWS server. To install a new font, you must run `bldfamily(1)`.

Adding F3 Fonts

The new font should consist of a font file (for example, `newfont.f3b`), a map file (for example, `Special.map`), and possibly some hand-tuned bitmaps (for example, `NewFont10.fb` and `NewFont12.fb`). If the new font uses one of three standard encodings (Latin, Dingbats, or Symbol), the map file is unnecessary.

You can install a new font anywhere in the file system. If you have no special need, it is simplest to use the default font directory. You must run `bldfamily(1)` with the pathname of the directory where the new font is located.

```
example% bldfamily -d fontdirectory
```

Remember that if your new font resides somewhere other than the default font directory, you must set the value of `FONTPATH` to include the new directory. (See the section *Locating Fonts*.)

To install the map file, use the program `map2ps(1)` to create entries in two directories: the *EncodingDirectory* and the *ExternalEncodingDirectory*. `map2ps` produces a file called `fontmaps.ps`, which you must load into the X11/NeWS server before using any of the new fonts.

```
example% map2ps mapfile
```

Generating Bitmap Files

You may want to build a bitmap version of an outline font. The bitmap version loads faster, since the server does not need to scale it, and the ASCII format file can be ported to other servers.

You can use the utility `makeafb(1)` to produce bitmap fonts from F3 outline fonts. For example, if you want to produce a 32-point Times-Roman bitmap font file, execute the following command:

```
example% makeafb -32 $OPENWINHOME/lib/fonts/Times-Roman.f3b
# This command produces a TmsR32.fb file and
# a TmR.afm file. These files can be used by
# the X11/NeWS server as a normal bitmap font.
```

You can generate bitmap fonts for specific sizes of an F3 font using the `makeafb(1)` program. (See the `makeafb(1)` manual page for more information.) Once a bitmap file is created, you need to run `bldfamily(1)` to update the information on this font family and `convertfont(1)` to convert the map file.

For example, to generate bitmap files for the font `shapefont.f3b`, which uses the non-standard map file `Shapes.map`, you run the following sequence of programs:

```
example% makeafb -e Shape.map shapefont.f3b
example% convertfont -e Shape.map
example% map2ps Shape.map
```

5.6. Converting NeWS 1.1 Fonts for Use with the X11/NEWS Server

The binary format changed between NeWS 1.1 and X11/NEWS; therefore, you need to update your private fonts. Use `convertfont` to convert old `.fb` files into the current version.

5.7. Font Aliases

Fonts can be aliased in your `.user.ps` file, as follows:

```
FontDirectory begin
  /myname      /systemname    _FontDirectorySYN
  /myname2     /systemname2   _FontDirectorySYN
  ...
end
```

The names in *systemname* must be existing fonts. They must also be the true name of the font — the form `<name>-<pointsize>` does not work.

NOTE *X11 defines font names to be case insensitive. To solve case problems, the font names are all converted to lower case by the server when it tries to find them in the FontDirectory. Since the above example is actually adding new keys (which are case sensitive) to the FontDirectory dictionary, any name to be used from X11 must be in lowercase, or the X11 code will be unable to find it.*

5.8. Fonts in the X11 Window System

The following sections describe how fonts are handled in X11 window system.

Using OpenFonts with the X11 Protocol

X11 can access the OpenFonts scaling technology through an enhanced version of the X Logical Font Description (X LFD) protocol. The X LFD gives numeric values for pixel size, point size, resolution, and maximum width. Since these variables have an indefinite range of values for a scalable font, these fonts are reported (through `xlsfonts` or the `XListFonts()` request) as having 0 in these fields. To use them, take the reported name and replace the pixel size or point size field with the desired value. Note that the point size argument is in decipoints, and it takes precedence if both values are supplied.

NOTE *For the server to parse these values correctly, the name must be supplied with all the dashes intact. Using wildcards to leave out sections of the name fails, though wildcards are safe to use for individual fields in the name.*

Alternatively, the fonts can be accessed by their POSTSCRIPT language names, with the point size appended to the name (for example, `Times-Roman-23`). This method works for any NeWS font, though scaling happens only for those fonts that can be scaled. Note that this is not a portable way of specifying fonts and should

be avoided when writing portable code. This font specification is much easier to type and remember than an X LFD format and may be preferable as a command line or `.Xdefaults` value.

X11 Access to NeWS Fonts

X11 can use any font in the font dictionary. The protocol requests `XListFonts` and `XListFontsWithInfo` return the list of currently installed fonts.

The protocol requests attempt to return all available fonts. However, in `OpenWindows`, the presence of infinitely scalable NeWS fonts means that the number of available fonts is itself infinite. Therefore, the X11/NeWS server responds to the requests by returning a list that contains all bitmap fonts (including all X fonts) and all `OpenFonts`.

X11 specifies that font names are ISO-Latin 1, case independent. NeWS, however, uses mixed capitals in font names. Therefore, in the font dictionary, font synonyms are used for NeWS fonts: each NeWS font has a lowercase symbol pointing to the real font.

5.9. Font Limitations

The following fonts are included for compatibility with X11, `SunView`, and `XView`. They cannot be rotated or scaled, and thus should not be used from NeWS applications.

```
Screen
Screen-Bold
Charter-Black
Charter-Black-Italic
Charter-Italic
Charter-Roman
fixed
9x15
8x13
8x13bold
```

Color Support

Color Support	57
Color in the NeWS Window System	57
Color in the X11 Window System	57
6.1. X11/NeWS Visuals and Colormaps	57
X11/NeWS Visuals	58
6.2. Recommendations	59
X11 Programmers	59
NeWS Programmers	60
6.3. NeWS Dynamic Colors	60



Color Support

This chapter is an introduction to the color support provided by the X11/NeWS server. It describes the differences between the color-handling techniques of X11 and NeWS window systems; it also makes recommendations on methods of programming in color.

This chapter assumes the reader's familiarity with the principles of color window systems. For information on color-related types and operators provided by NeWS, see the *NeWS Programmer's Guide*. For information on X11-specific methods of color support, see the appropriate X11 documentation.

Color in the NeWS Window System

In NeWS, each color can be specified as either a red/green/blue or a hue/saturation/brightness triplet. (On grayscale terminals, the standard NTSC triplet values of .56/.33/.11 are used for every shade; variation is produced by increasing or decreasing the brightness.)

When an application requests a particular color, the server attempts to match the specified color with the colors available on the current hardware. Only two colors, black and white, are available on a monochrome server. The server uses techniques such as *dithering* and *color substitution* to make the best possible approximation of the requested colors. Once a color has been chosen and used, the application can call the `contrastswithcurrent` operator to determine the similarity of the used color to the requested color.

The graphics context supports the concept of a *current color*; each specified drawing operation is automatically performed in the current color.

Color in the X11 Window System

The X11 protocol can be used with several different classes of display device; it allows applications to treat each class of device differently. Thus, if applications need to use color, they should query the server to discover the methods of color-display that it supports; otherwise, calls made to routines that use colors may fail. The queries required of X11 applications that use the X11/NeWS server are described throughout this chapter.

6.1. X11/NeWS Visuals and Colormaps

A hardware *colormap* is a color lookup table that determines which color is displayed for a specified pixel value. Colormaps can be of two kinds:

- A *static* colormap
This contains fixed color values that cannot be changed.
- A *dynamic* colormap
This contains values whose color-correspondence can be modified by an application.

X11 applications sometimes require access to a dynamic colormap; existing NeWS applications should never require such access. The X11/NeWS color model is therefore created to satisfy the requirements of both protocols.

X11/NeWS Visuals

Most color framebuffers supported by X11/NeWS have 8 bits per pixel; thus, each pixel has 256 possible colors. Each pixel value is associated with a color according to the colormap; the colormap contains 256 slots, one for each color.

The correspondence of a canvas to the color mappings contained in a hardware colormap is controlled by a system-defined facility known as a *visual*. The X11/NeWS server currently uses two visuals for the 8-bit color framebuffer:

- The `StaticColor` visual
This visual, which in X11 terminology is the *default visual*, is automatically used by all NeWS clients. It is associated with a fixed colormap that uses only 240 colors. All X11 applications using `XAllocColor` on the default color map also use this visual and, like NeWS clients, are given the best available match to the color they request.
- The `PseudoColor` visual
This visual must be used by X11 clients that perform colormap-modification operations. It is associated with a colormap that contains all 256 available colors; the first 16 color values required by the client are automatically associated with the 16 slots not used by the `StaticColor` default visual.

Note, however, that if one or more X11 clients together require more than 16 colors, additional slots are reclaimed from the 240 slots in the default visual; in this situation, the colors currently displayed by NeWS and X11 applications may change unpredictably.

This configuration allows X11 applications that perform colormap modifications to run simultaneously with NeWS and X11 applications that do not. The configuration favors applications that are able and willing to use colors that are very close to the ones desired.

Note that many X11 applications do not anticipate the availability of more than one visual; thus, they simply use the default visual and call `XDisplayCells()` to determine its depth (a depth greater than two indicating the availability of color). However, in the context of the X11/NeWS server, this procedure should not be relied on. It might lead to an X11 application accidentally attempting colormap modifications on the default `StaticColor` visual, which results in error.

Note also that if an X11 client program ascertains the availability of color and then attempts to manipulate colors with calls such as `XAllocColorCells()`, the program crashes, since the default visual for color X11/NEWS servers is `StaticColor`: client programs wishing to manipulate the colormap must access the dynamic visual, which is `PseudoColor`.

NOTE *On monochrome machines (which usually have the `bwtwo(4S)` framebuffer), an attempt to return the available visuals by calling `XGetVisualInfo()` returns only a two-color `StaticGray` visual.*

6.2. Recommendations

This section contains recommendations for programmers who intend to use the X11/NEWS color support facilities.

X11 Programmers

If you are programming in X11, proceed as follows:

- Do not attempt to use a visual before you have checked on all supported visual types, using `XGetVisualInfo()`, and have checked for the best match obtainable from these, using `XMatchVisualInfo()`.
- Examine your application and ascertain whether you really need to manipulate your own colormap. If you do not, you should use the default `StaticColor` visual.
- If you know exactly which colors you need, use `XAllocColor()` and `XAllocNamedColor()` to obtain the closest possible match from colors in the server's fixed colormap. (Note that these routines can also be used in the context of the `PseudoColor` visual.)
- If you need to manipulate colors with the `PseudoColor` visual, try to manipulate as few colors as possible, thereby avoiding the chance of an overlap occurring between the `StaticColor` and `PseudoColor` visuals.
- Do not access the `lib/rgb.*` files used by some X11 implementations: these files are superfluous in X11/NEWS, since X11-named colors (such as "aquamarine" and "cornflowerblue") are looked up in the same `ColorDict` POSTSCRIPT language dictionary used by NEWS, which is located in `$OPENWINHOME/etc/NEWS/colors.ps`.

NOTE *The X11 protocol specifies that color names are case-insensitive; thus, any color name specified by an X11 application is converted into lowercase before it is looked up. Therefore, if you add your own color names to `ColorDict`, make them lowercase. For a more detailed treatment of this subject, see *Visualizing X11 Clients*.²⁰*

²⁰ David Lemke and David S. H. Rosenthal, *Visualizing X11 Clients*, pp 125-138, USENIX Conference, San Diego, CA, February 1989.

NeWS Programmers

If you are programming in NeWS, proceed as follows:

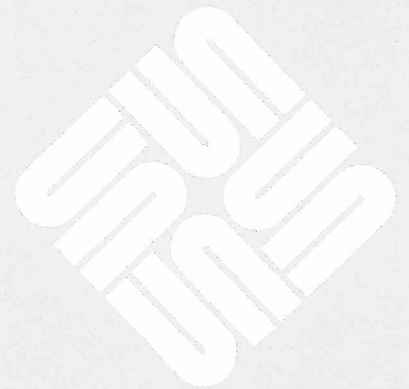
- Use the named colors in **ColorDict** in preference to user-defined *rgb* or *hsb* shades; the named colors are likely to be closer to the actual shades produced on the monitor and will not require dithering.
- Do not rely on XOR (**setrasteropcode**) to provide complementary colors. 16 slots of the colormap contain values that will be changed by certain X11 applications; thus, the complements of these values will also change unpredictably.

6.3. NeWS Dynamic Colors

NeWS supports the same dynamic color operations as X11. For further information, see the *NeWS Programmer's Guide*.

Using SunView Windows

Using SunView Windows	63
Running SunView Applications	63
Bugs in SunView/X11/NeWS Coexistence	64
Inconveniences	64
Screen Damage	64
Input Mismatches	64



Using SunView Windows

You can run unmodified SunView (or SunWindows-based) binaries on the X11/NeWS server; thus, SunView windows can be displayed on the screen simultaneously with NeWS and X11 windows.

Note that when multiple windows are used in this way and are made to overlap, the SunView windows always obscure the windows and menus generated by X11 or NeWS programs. (This occurs because the root window is managed by X11/NeWS rather than by the SunView programs; the SunView programs effectively update the display without reference to the X11/NeWS windows.) Each SunView window is surrounded by a white margin; this prevents leftover cursor images from appearing when the cursor is moved between the SunView window and the X11/NeWS environment.

NOTE *Running SunView tools under X11/NeWS when X11/NeWS has been initialized from within SunView is not supported on color displays.*

If you run X11/NeWS, note the following:

- Multiple SunView applications may run at the same time.
- SunView cross-hairs function and appear correctly.
- *Cut and paste* selections can be exchanged between X11/NeWS and SunView programs. However, secondary selections between the two environments should not be attempted.
- Old versions (back to 1.1) of SunWindows-based applications work as well as they would under SunView.

Running SunView Applications

To start a SunView application, type the application's full pathname. SunView applications are stored in `/usr/bin`. Thus, the following line starts the SunView mailtool:

```
example% /usr/bin/mailtool
```

Bugs in SunView/X11/NEWS Coexistence

This ability to run SunView programs from X11/NEWS is useful but not flawless. The following subsections describe the problems that exist.

Inconveniences

- A color SunView application needs the cursor over its window in order to see the application's true colors.
- Annoying messages, such as the following, may appear on the console:

```
Window display lock broken after time limit exceeded \
by pid nnn
```

To reduce the occurrence of this problem, adjust the display lock timeout by modifying the kernel with the `adb(1)` command; see Section 7.5, *Kernel Tuning Options*, in the *SunView System Programmer's Guide*.

Screen Damage

Screen damage sometimes (though rarely) occurs when you open a SunView application over the cursor that is on X11/NEWS' part of the screen. To repair the damage, use the Refresh menu command.

Input Mismatches

SunView sets up the `kb(4S)` keyboard driver in the kernel to deliver encoded events, while X11/NEWS uses an unencoded keyboard. The X11/NEWS server resets the keyboard state to SunView encoding when the keyboard focus moves into a SunView window and when X11/NEWS exits; therefore, you should not notice the difference. However if X11/NEWS hangs for some reason, you may be left with the keyboard producing random characters in SunView windows. The program `kbd_mode` switches the keyboard between the different modes; you can login remotely to your machine and type `kbd_mode -e` to reset to SunView mode.

Since the keyboard state is changed when you change the keyboard focus between X11/NEWS and SunView, do not hold any keys down when you move the mouse from one environment to the other; the environment you were in to begin with never sees the key going up, and thus is confused about the keyboard's state when you re-enter it. This may leave you in secondary selection mode. You should be able to clear this by pressing the **Stop** key twice (usually **LI** in X11/NEWS or SunView).

If this does not reset the state of the function keys, you can login remotely to your machine and type `clear_functions(1)` to get SunView's selection mechanism out of a constant secondary selection mode.

If you have SunView and X11/NEWS running on separate displays, you can invoke these operations more conveniently if you add **Reset Selection** clear functions to your SunView `rootmenu` file. For example:

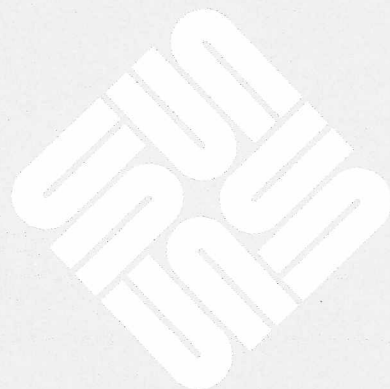
```
"Reset Keyboard" kbd_mode -e
```

Another problem can occur if a SunView program enters *fullscreen access* mode to prompt the user for keyboard input when the keyboard mode is still set to unencoded. An example of this is an application prompting for a password before allowing the application to run. A workaround for this is to make sure that a SunView application has the keyboard focus before invoking an operation that requires a prompt.

A

NeWS Manual Pages

NeWS Manual Pages	65
24to8.1	67
bldfamily.1	69
buildmenu.1	71
convertfont.1	73
cps.1	75
journalling.1	77
kbd_mode.1	79
ldf.1	81
makeafb.1	83
mkiconfont.1	85
newshost.1	89
newsserverstr.1	91
objectdiff.1	93
objectwatcher.1	95
openwindemos.6	97
pageview.1	103
pam.1	105
psh.1	107
psindent.1	109
psio.3	111
psman.1	117
psps.1	119



NAME

`24to8` – convert a 24 bit rasterfile to an 8 bit one suitable for X11/NeWS.

SYNOPSIS

`24to8` [`-v`] [`-q`] [`inraster` | `-`] [`outraster`]

DESCRIPTION

`24to8` takes as input a 24 bit Sun rasterfile(5) and reduces the depth of the image, from truecolor to 8 bit colormapped index color. `24to8` uses Floyd/Stienberg dithering to achieve high quality images while maintaining the X11/NeWS static colormap.

If both filenames are missing, the source rasterfile is read from `stdin` and the output rasterfile is written to `stdout`. If there is only one filename, then it is interpreted as the input rasterfile. To have a named output rasterfile and still read the input rasterfile from `stdin`, use a dash (`-`) in place of the input filename.

OPTIONS

- `-v` Verbose mode will print information as it processes the image. (The default is to be silent.)
- `-q` Query (prints list of options)

BUGS

Floyd/Stienberg dithering could be bidirectional and use blue noise for better results.

SEE ALSO

`lpr(1)`, `8to24(1)`, `tga2ras(1)`, `rgb2ras(1)`, `ras2ps(1)`

AUTHOR

Patrick Naughton (naughton@wind.sun.com)

COPYRIGHT

Copyright (c) 1989 by Sun Microsystems, Inc.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.



NAME

bldfamily – build font family description

SYNOPSIS

bldfamily [**-d** *dirname*] [**-f** *n*]

DESCRIPTION

bldfamily scans *dirname* for NeWS font files (files with extensions **.fb** and **.fm**) and constructs a NeWS font family file for each group of font files with the same family name. **bldfamily** also creates a file named **Families.list** that contains a list of all the font families in the directory. If *dirname* is not specified, it defaults to **\$FONTDIR** if defined, to **'.'** otherwise. Each family file that is built is given the family name followed by the suffix **.ff**.

A font family is a set of font files that are grouped together to provide a single POSTSCRIPT font. In the POSTSCRIPT language, each font has a name, such as **Times-Roman**, and can be rendered in many different sizes. A NeWS font file is an instance of a POSTSCRIPT font at a particular size. Font family files contain the information necessary for NeWS to pick the right bitmap font.

OPTIONS

-d *dirname* Specifies the directory to scan and put the **.ff** file into.

-f *n* Sets the maximum length of an output filename (excluding extension) to *n*. When writing NeWS format files, NeWS normally constructs the output filename from the name of the font and its scaling factors. Some systems cannot cope with long file names, so this option can be used to squeeze the name heuristically. The default value is 8.

EXAMPLE

example% bldfamily -d /usr/newfonts

In this example, **bldfamily** scans **/usr/newfonts** and builds a font family file called **/usr/newfonts/Boston.ff**.

SEE ALSO

convertfont(1)

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc.

NAME

buildmenu - builds an OpenWindows (tNt) Menu from a SunView style menu file.

SYNOPSIS

buildmenu [**-root**] [**-title** *title*] [**-position** *position*] [**-pinnable**] [*menufile*]

DESCRIPTION

buildmenu converts a SunView menu file into the NeWS toolkit PostScript commands to build the same menu under OpenWindows.

If the filename is missing, the menufile is read from `~/openwin-menu`. If the filename is '-', the menufile is read from stdin. In any case, the PostScript language is written to stdout.

The expected usage would be for the user to *insert* a submenu into the default root menu rather than simply copying the default root menu and modifying it.

For example if you have a SunView style menu in `~/mymenu` you would put this line in your `.openwin-init` to append your menu to the default OpenWindows root menu.

buildmenu `~/mymenu -position 99 -title "My Tools" -pinnable | psh`

The OpenWindows Menu File Format

The file `$OPENWINHOME/lib/openwin-menu` contains the specification of the default OpenWindows menu. You can change the OpenWindows menu by creating your own file and storing it in your home directory with the name `.openwin-menu`.

Lines in the file have the following format: The left side is a menu item to be displayed, and the right side is a command to be executed when that menu item is chosen. Lines may be continued to the next line by putting a backslash at the end of the line to be continued. Blank lines and comment lines beginning with '#' are ignored.

The menu items can be any string as long as strings with embedded blanks are delimited by double quotes. The SunView notion of having the full pathname of an icon file delimited by angle brackets is unsupported and the `basename(1)` of the icon file is used as the item label in place of the icon.

There are five reserved-word commands that can appear on the right side.

- EXIT** Exit OpenWindows (requires confirmation).
- REFRESH** Redraw the entire screen.
- MENU** This menu item is a pull-right item with a submenu. If a filename follows the MENU command, the submenu contents are taken from that file, (shell variables and '~'s are interpreted correctly). Otherwise, all the lines between a MENU command and a matching END command are added to the submenu.
- END** Mark the end of a nested submenu. The left side of this line should match the left side of a line with a MENU command. If the string PIN follows the END command then this submenu has an OpenLook pin on it.
- POSTSCRIPT** This rest of the line is to be directly interpreted as PostScript language code when the item is invoked.

If the command is not one of these five reserved-word commands, it is treated as a command line and executed. Full shell interpretation is done on the commands.

Here is a menu file that demonstrates all of these features

```
"Mail reader"       mailtool
"My tools"           MENU               ~/rootmenu
"Click to type"     POSTSCRIPT       /ClickFocus setfocusmode
"Follow mouse"     POSTSCRIPT       /CursorFocus setfocusmode
"Print selection"   news_selection | lpr
```

"Nested menu"	MENU	
	"Command Tool"	cmdtool
	"Shell Tool"	shelltool
"Nested menu"	END	PIN
Repaint	REFRESH	
Quit	EXIT	

OPTIONS

-root

This replaces the existing rootmenu rather than inserting or changing an item in it. OpenWindows uses this mode to install your default menu from `~/openwin-menu` or `$OPENWINHOME/lib/openwin-menu` if `~/openwin-menu` does not exist.

-title *title*

This sets the name of the menu item you wish to insert. If there already exists an item in the root menu with the label "*title*" then this menu item is replaced by this menu. If you have also set **-root** above then this sets the title of the root menu.

-position *position*

This sets where you would like this item to be inserted in the root menu. This option has no effect if you specify **-root**, since you are replacing the whole root menu. The default is 0, meaning insert at the beginning of the root menu.

-pinnable

This makes the top level menu you are inserting have an OpenLook pin. The default is to not have one.

FILES

`$OPENWINHOME/lib/openwin-menu`

SEE ALSO

`xnews(1)`, `sunview(1)`

NAME

convertfont - dump font out in some other format

SYNOPSIS

convertfont [-a | -b | -v | -vf | -x] [-c *comment*] [-d *dirname*] [-f *n*] [-n *fontname*] [-S] [-s *n*] [-t] [-tv] [-ta] *filenames*

DESCRIPTION

convertfont reads in the set of named font files and dumps them out again according to the specified options, effectively converting the files from one font format to another. convertfont is typically used to generate fonts for use with the X11/NEWS window system.

There are five types of font file that convertfont can read: Sun standard vfont format, Adobe ASCII bitmap format, Adobe ASCII metric format, NEWS font format, and CMU (Andrew) format. The format of the input font is determined automatically by inspecting the file. It can write fonts out in one of three formats: Adobe ASCII, NEWS, and vfont. The default output format is NEWS.

OPTIONS

- a Selects Adobe ASCII output format. This is the format that you should use when transporting fonts from one machine architecture to another. The output file extension will be .afb or .afm.
- b Selects NEWS output format (the default). The output file extension will be .fb. If the input file is an Adobe ASCII metrics file, the extension will be .fm.
- v Selects vfont output format. The output file extension will be .vft.
- vf Selects vfont output format. The output file extension will be .vft. Forces the characters to be fixed width.
- c *comment* Sets the *comment* field of the font. The Adobe ASCII and NEWS font formats support an internal comment that accompanies the font. This is usually used to contain copyright or history information. It is normally propagated automatically.
- d *dirname* Specifies the directory into which the font files will be written. If the FONTDIR environment variable is set, it is used as the default value. Otherwise, if the OPENWINHOME environment variable is set, \$OPENWINHOME/fonts is used as the default value. Otherwise '.' is used.
- f *n* Sets the maximum length of an output filename (excluding extension) to *n*. When writing NEWS format files, NEWS normally constructs the output filename from the name of the font and its scaling factors. Some systems cannot cope with long file names, so this option can be used to squeeze the name heuristically. The default value is 8.
- n *name* Forces the output font name to be *name*. It is important to not confuse the name of the font with the name of the file that contains it. Some font formats (Adobe ASCII and NEWS) contain the name of the font internally. So, given a 10-point Times-Roman font, its font name will be Times-Roman, but its file name might be TmsR10.fb.
- S Attempts to determine the size information of fonts by inspecting the bitmaps and applying some heuristics. This is useful when reading vfonts (particularly those intended for printers like the Versatec) that are missing or have incorrect size information.

- s *n*** Sets the point size of the font to *n*. Overrides any internal size specification.
- t** Prints a short description of the fonts on standard output; a reformatted font file is not dumped.
- tv** Prints a more verbose description of the fonts on standard output; a reformatted font file is not dumped.
- ta** Prints a long description of the fonts on standard output; a reformatted font file is not dumped.
- x** Selects Adobe/MIT X11 BDF 2.1 output format. This is the format that you should use when transporting fonts between X11 servers. The output file extension will be **.bdf**.

SEE ALSO**bldfamily(1), vfont(5)****DIAGNOSTICS**

- Bad flag: -C** Unknown command line option
- Couldn't write ...** Error writing font file
- f*: not a valid font.** Unknown input file format

NAME

`cps` – construct C to POSTSCRIPT language interface

SYNOPSIS

`cps` [`-c`] [`-D symbol`] [`-I filename`] [`-s [number]`] [`-i`] [`file.cps`]

DESCRIPTION

`cps` compiles a specification file containing C procedure names and POSTSCRIPT language code into a header file (`file.h`) that can be included in C programs. The header file associates the C procedure names with macros that transmit a compressed form of the POSTSCRIPT language code to the X11/NEWS server stream. Only one input file can be specified. If the `file.h` file already exists, a backup copy of it will be generated of the form `file.h.BAK` before the new file is generated.

The convention is for the input specification file to end with the suffix `.cps`.

OPTIONS

`-c` Compiles the file of POSTSCRIPT language code for faster loading by NEWS, but does not generate a specification file for C programs. For example, the command line

```
example% cps -c < input_file > output_file
```

converts the input file from the ASCII form of the POSTSCRIPT language to a compressed binary form. When read by NEWS, the output file will execute exactly the same as `input_file`, except that it will be parsed faster.

NOTE: The `-c` option will not work if the input file uses constructs such as `currentfile readstring`, which are often used with the `image` primitive.

`-C` Compiles the file of POSTSCRIPT language code in the same way that the `-c` option does: however, when `-C` is used, the file can contain usertoken specifications. The tokens are automatically set up at the start of the output file; they are used throughout the output file to compress the POSTSCRIPT language even further than occurs with the `-c` option.

`-D symbol` Defines symbols to be passed to the C language preprocessor (`cpp(1)`), which processes the input file.

`-I filename` Specifies include files or include paths. Passed on to the C preprocessor.

`-i` Generates two specification files: the first file contains only the compressed form of the POSTSCRIPT language code; the second file contains the macro definitions required for the C-POSTSCRIPT interface. For example, `ps_open_PostScript()` and `ps_close_PostScript()` would be defined in the second file. The second file references compressed POSTSCRIPT language code as `extern char` arrays. The first file is of the form `file.c`; the second file is of the form `file.h`. If the files already exist, `.BAK` backup copies will be generated.

This option is useful for minimizing the size impact of CPS interfaces that contain procedures called from several places in the C code. The `file.c` generated would only need to be compiled once. Each file that needs to use the interface could then include only the `file.h` and use the macros in that file multiple times. Each repeated invocation of the macro would refer to the shared POSTSCRIPT language code in the `file.c` rather than its own static copy of the POSTSCRIPT language code.

`-s [number]`

Specifies the threshold at which compiled POSTSCRIPT language code will be output as decimal arrays instead of string constants. If `number` is missing, all POSTSCRIPT language code will be output as string constants in the resulting `file.h`. This may be useful for debugging purposes, even though the POSTSCRIPT language code is in compressed form. If `number` is 0, all POSTSCRIPT language code will be output as decimal arrays. The default threshold is 400 characters, which is less than the maximum limit of string constants for most compilers. Note that there must be no space before `number`, since it is optional.

SEE ALSO

cpp(1V)

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems, Inc.

NAME

journalling – NeWS event record and playback package

SYNOPSIS

journalling

DESCRIPTION

The Journalling package allows you to capture NeWS mouse and keyboard events onto a file and play the file back. This results in NeWS' faithfully duplicating the original user actions in real time.

This package permits continuous replaying of a given file. Playback can be interrupted at any time by clicking on the **Interrupt** button.

USAGE

Invoke the journalling program from the **Demo Navigator**: a few seconds are required for journalling to initialize. When initialization is complete, a control panel window appears. The control panel contains the following items:

RECORD, STOP, and PLAY buttons — Pressing **RECORD** starts recording onto the current recording file. Pressing **STOP** terminates recording. Pressing **PLAY** starts playback from the current playback file. The buttons light up to indicate what action is currently taking place.

Record File — This text item allows you to specify the current file onto which action items are recorded. This can be any valid file on the server machine. Relative pathnames are taken to be relative to the directory from which NeWS was started. The default for the **Record File** is `/tmp/NeWS.journal`.

Playback File — This text item allows you to specify the current file from which recorded action items are played back. This can be any valid file on the server machine. Relative pathnames are taken to be relative to the directory from which NeWS was started. The default for the **Playback File** is `/tmp/NeWS.journal`.

Play Forever toggle switch — If this switch is on, the action items in the **Playback File** are played back repeatedly without stopping. The playback can be terminated with the **Interrupt** button (see below).

Done button — When the mouse is clicked on this button, all journalling items are removed from the server, and the control window is unmapped. (This is equivalent to selecting **Quit** from the frame menu.)

Interrupt button — When the mouse is clicked on this button, playback is interrupted. The button flashes when selected.

TIPS FOR USING JOURNALLING

When a journal is created for repeated playback, all windows created after journalling has started should be removed before journalling ends: otherwise, the server re-creates the windows whenever playback begins and eventually runs out of memory. Thus, at the end of a journalling session, the state of the screen should be exactly as it was at the beginning.

Different machines produce a noticeable variation in journalling performance. For example, journalling is faster on a Sun 4 than on a Sun 3/50. Playback of a script recorded on a fast machine might not work correctly on a slower machine. Any given machine has a maximum rate at which NeWS events can be handled.

Care must be taken when recording sequences that contain invocations of Unix programs, particularly when starting new applications. The mouse must not be clicked until the bounding box is up on the screen.

If the mouse is clicked early, the wrong window-sizing will be made on playback: this will lead to unpredictable behavior, due to the window's not being where it was when recording.

Always proceed slowly while recording a script. Remember there is no synchronization. For example, when recording a sequence of actions such as typing 'ls -l' into a terminal window and then cutting and pasting the command into another terminal window, allow the command to complete before cutting and pasting: otherwise indeterminate results may follow on playback.

FILES

`${OPENWINHOME}/demo/journalling`

BUGS

Do not use the Journalling package for critical functions: the behavior of playback is unpredictable, due to the non-deterministic nature of the Unix scheduling mechanism and the general operating environment.

NAME

kbd_mode – change the keyboard translation mode

SYNOPSIS

kbd_mode **-a|-n|-e|-u**

DESCRIPTION

kbd_mode sets the translation mode of the console's keyboard (`/dev/kbd`) to one of the four values defined for `KIOCTRANS` in `kb(4S)`. This is useful when a program that resets the translation mode crashes; for example, `NeWS` (when run from `SunView`) can sometimes leave `SunView` reading untranslated events.

Note that `SunView` desires translated events (**kbd_mode -e**), while `X11/NeWS` desires untranslated events (**kbd_mode -u**). See below for an explanation of the **-e** and **-u** options.

OPTIONS

- a** ASCII: the keyboard will generate simple ASCII characters.
- n** None: the keyboard will generate unencoded bytes – a distinct value for up and down on each switch on the keyboard.
- e** Events: the keyboard will generate SunWindows input events with ASCII characters in the *value* field.
- u** Unencoded: the keyboard will generate SunWindows input events with unencoded bytes in the *value* field (this is the mode `NeWS` currently uses).

FILES

`/dev/kbd`
`$OPENWINHOME/bin/kbd_mode`

SEE ALSO

`kb(4S)`

KBD_MODE(1)

USER COMMANDS

KBD_MODE(1)

NAME

ldf – load POSTSCRIPT-defined News font

SYNOPSIS

ldf [*fontname*]

DESCRIPTION

ldf loads a font into the server. When no argument is specified, it prints out all the POSTSCRIPT language fonts in \$FONTPATH. When an argument is specified, **ldf** searches in \$FONTPATH for a font with that name and loads the first instance that it finds.

LDF(1)

USER COMMANDS

LDF(1)

NAME

makeafb – create bitmap files from scalable Folio format files

SYNOPSIS

makeafb [**-m|-M**] [**-p|-P**] [**-v|-V**] [**-f n**] [**-values**]

DESCRIPTION

Makeafb creates Adobe ASCII format bitmap fonts (*.afb*) and Adobe ASCII format metric files (*.afm*) format files from scalable Folio format files. These can then be converted with *convertfont(1)* for the X11/NeWS server. This is useful to avoid the calculation overhead of standard sizes of fonts.

OPTIONS

- m** Enable generation of *.afm* files (the default).
- M** Disable generation of *.afm* files.
- P** Don't preserve existing files (the default).
- p** Preserve existing files. If **-p** is selected, then just before *makeafb* writes a file it will check to see if it already exists. If it does, the file will be skipped. This is useful in situations where you have some handbuilt *.afb* and *.afm* files, and just want to fill in the missing ones.
- v** Verbose: print messages indicating what's going on (the default).
- V** Work silently.
- f n** Force the length of the base part of the output filename to be at most *n* characters. The default is 8.
- values** A comma separated list of pixel sizes for which *.afb* files should be generated. The default is *6,8,10,12,14,16,18* .

EXAMPLES

makeafb *.f3b

makeafb -p -4,5,6,7,8,9,10,11,12,14,16,18,20,24 *.f3b

SEE ALSO

convertfont(1), *bldfamily(1)*

NAME

mkiconfont — make an ASCII cursor or icon font from a list of ASCII bitmap files

SYNOPSIS

mkiconfont [*listfilename*] [*fontname* > *filename.afb*]

DESCRIPTION

mkiconfont makes an ASCII version of the font *fontname* from the ASCII bitmap files that are listed in *listfilename* and puts the output in the file *filename.afb*. The convention is to use the suffix *.afb* for the output file. The ASCII bitmap files must conform to a specific format.

mkiconfont is used to create cursor fonts and icon fonts. Each cursor has a cursor image and a mask image that are superimposed to create the complete cursor. To create a cursor font, first make a bitmap file for the cursor image and another bitmap file for the mask image. Then make a list of your cursor image and mask image bitmap files, and save your list as *listfilename*. Next run **mkiconfont**, and then run the output *filename.afb* through the **convertfont(1)** and **bldfamily(1)** utilities. Follow the same procedure to make an icon font, but omit the mask image files.

EXAMPLE

The font utility **mkiconfont** expects input in the format illustrated by the examples below. Here is an example of an ASCII bitmap file for a cursor image named **pointer**. Its image is that of a narrow arrow that points up and to the left.

```
/* Format_version=1, Width=16, Height=16, Depth=1
 * Valid_bits_per_item=16, XOrigin=0, YOrigin=15
 */
0x0000,0x4000,0x6000,0x7000,0x7800,0x7C00,0x7E00,0x7800,
0x4C00,0x0C00,0x0600,0x0600,0x0300,0x0300,0x0180,0x0000
```

XOrigin and **YOrigin** indicate the origin of the character, which is the hot-spot of the cursor. The values for **XOrigin** and **YOrigin** originate in the bitmap's lower left corner with positive values extending up and to the right. Note that **YOrigin** starts from the last non-zero row of pixels rather than from the bottom of the bitmap.

Here is the ASCII bitmap file for the mask image of the **pointer** cursor. It is called **pointer_mask**.

```
/* Format_version=1, Width=16, Height=16, Depth=1
 * Valid_bits_per_item=16, XOrigin=0, YOrigin=16
 */
0xC000,0xE000,0xF000,0xF800,0xFC00,0xFE00,0xFF00,0xFF80,
0xFE00,0xDF00,0x9F00,0x0F80,0x0F80,0x07C0,0x07C0,0x03C0
```

Note that the mask image is used to outline the primary image, and therefore its origin is offset by one from the primary image, so as to superimpose the images correctly. This arrangement is typical of cursor masks.

Here is the process for generating a simple cursor font (the process is the same for generating an icon font, except that no mask images are needed):

- 1) Generate a collection of ASCII bitmap file pairs with the format described above. The convention is to call each cursor image *name.cursor* and its mask image *name_mask.cursor*. Create a file containing these filenames, with each name on a separate line. The pair order should be the cursor image filename on one line followed by the mask image filename on the next line. You can give your list any filename. In this example, the file is called **myfont.list**.
- 2) Make an ASCII version of the font from the list of ASCII bitmap files using the program **mkiconfont**. The first argument to **mkiconfont** is the name of the file that contains the list of filenames. The second argument to **mkiconfont** is the name of the output file prepended by a > and the intended name of the font family.

example% mkiconfont myfont.list MyFont>MyFont12.afb

- 3) Convert the ASCII version of the font to a binary version using the program **convertfont(1)**. The first argument should be a **-d** flag followed by the directory in which you want to put the resulting binary font file. You will want to put your output font file in your font directory; in this example, the font directory is **\$OPENWINHOME/lib/fonts**. The next argument is the name of the file that contains the ASCII version of the font. **convertfont** names the output file like the ASCII version, but it uses a **.fb** suffix instead of a **.afb** suffix. In this example, the ASCII version is in the file called **MyFont12.afb**, and the output file that **convertfont** produces is called **MyFont12.fb**.

example% convertfont -d\$OPENWINHOME/lib/fonts MyFont12.afb

- 4) Build a font family file for the font, using the program **bldfamily(1)**. The only argument to **bldfamily** is the name of the directory in which the font files are located. **bldfamily** looks in the specified directory for files with extensions **.fb** and **.fm** and constructs a **NeWS** font family file for each group of fonts with the same family name.

example% bldfamily -d\$OPENWINHOME/lib/fonts

- 5) Create a **.ps** file that contains a dictionary of character names for the font. The **.afb** and **.fb** files associate a number with each character in the font; it is more convenient to associate the name of each bitmap file with the character that it represents. The following example shows one way to build such a **.ps** file.

```
#!/bin/sh
egrep "^(STARTCHAR|ENCODING)" MyFont12.afb>myfont.ps
ed - myfont.ps<<'EOFT'
g/STARTCHAR/j
1,$s'STARTCHAR *\(.*)ENCODING *\(.*)'/\1 /\2 def'
li
/myfontdict 300 dict def
myfontdict begin
.
$a
end
/showmyfont {
currentfont ( ) dup 0 myfontdict 5 index get put
myfontfont setfont show setfont pop } def
/myfont (MyFont) findfont 12 scalefont def
.
w
q
EOT
```

Another way to implement the name association is to have **mkiconfont** build the **.ps** file; this method is also a valid implementation.

- 6) Copy the **.ps** file to a well-known place.

example% cp myfont.ps \$OPENWINHOME/lib/NeWS

- 7) Use the **.ps** file before you use the font in your **POSTSCRIPT** program.

```
(NeWS/myfont.ps) run
myfontdict begin
name name_mask myfont newcursor
end
```

Note that your .ps file created the dictionary **myfontdict** for you. You can then push the dictionary on the stack and use it in the normal way.

SEE ALSO

bldfamily(1), convertfont(1)

FUTURE DIRECTIONS

In the future, **mkiconfont** will be replaced with a more sophisticated font editing tool. The new tool will be useful for creating any new font, rather than just being useful for fonts with a limited number of characters such as icons and cursors.

NAME

newshost – NeWS network security control

SYNOPSIS

newshost add [*hosts*]
or newshost remove [*hosts*]
or newshost show

DESCRIPTION

newshost is a shell command that manipulates the registry of hosts that are allowed to connect to the X11/NeWS server. The identity of the X11/NeWS server whose registry will be manipulated is determined by the **NEWSSERVER** environment variable. The variable **/NetSecurityWanted** (in the NeWS **systemdict**) may be set to **false** to disable the security mechanism.

newshost add	Adds the named hosts to the registry.
newshost remove	Removes the named hosts from the registry.
newshost show	Prints out a list of the hosts in the registry.

SEE ALSO

NeWS Programmer's Guide



NAME

newsserverstr — generate a string for the **NEWSERVER** environment variable

SYNOPSIS

newsserverstr [*hostname*] [*portnumber*]

DESCRIPTION

newsserverstr generates and prints the proper value of the **NEWSERVER** environment variable for *hostname* and *portnumber*. If **NEWSERVER** is then set to this value, **NEWS** clients will attempt to connect to *hostname* at *portnumber*. The default value for *hostname* is the current host, and the default value for *portnumber* is 2000. The two arguments can be specified in either order on the command line.

The format of the **NEWSERVER** environment variable is as follows:

decimal-address . port# ; hostname

For example, if the host called **myhost** has address **192.98.34.118**, the **NEWSERVER** variable could be set to **3227656822.2000;myhost** to enable **NEWS** clients to connect to the **X11/NEWS** server on **myhost** at port 2000.

newsserverstr simply calculates the string and sends it to standard output; you should then set the environment variable **NEWSERVER** to the value returned by **newsserverstr**.

EXAMPLE

C-shell users can define the following alias:

```
alias snh 'setenv NEWSERVER `newsserverstr !*`'
```

and Bourne Shell users can define the following function:

```
snh () {
    NEWSERVER='newsserverstr $*'
    export NEWSERVER
}
```

Both the above forms let you simply type '**snh hostname**' to set the **NEWSERVER** environment variable automatically.

SEE ALSO

newshost(1)

BUGS

If you use the **snh** alias or shell function, and the *hostname* you give is unknown, or you give too many or too few arguments, the **NEWSERVER** variable will be trashed.

NAME

objectdiff – list differences between two lists of X11/NeWS data objects

SYNOPSIS

objectdiff [*file1 file2*]

DESCRIPTION

objectdiff performs a **diff(1)** on two files containing output from the X11/NeWS operator **objectdump**. A formatted summary of the differences in the number and/or size of the objects of each type is output.

If the files were not produced by using **objectdump** on the same server during the same run, the output may be garbage. If the two files were produced by using **objectdump** on two different releases of the server, a change in accounting could cause synchronization problems.

Output is in the following format:

nnnnn bytes for *mmm* *object_type* objects

The output is sorted from the biggest space allocated at the top to the smallest space allocated or the biggest space freed at the bottom. At the end, there is a total-line of the following form:

nnnnn bytes for *mmm* *TOTAL* objects

The *object_types* are the same as those used for the output from **objectdump**.

EXAMPLES

To find the number of objects allocated by a given operation, use the following:

```
example% psh
executive
(/tmp/objects1) (w) file objectdump
/new MyClass send
(/tmp/objects2) (w) file objectdump
quit
example% objectdiff /tmp/objects1 /tmp/objects2
```

SEE ALSO

objectwatcher(1)

OBJECTDIFF(1)

USER COMMANDS

OBJECTDIFF(1)

NAME

objectwatcher – list data objects allocated/deallocated in X11/NeWS since this command was last run.

SYNOPSIS

objectwatcher

DESCRIPTION

objectwatcher is a Unix Bourne shell script that prints a formatted summary of data objects allocated and deallocated in X11/NeWS since the command was last run.

This tool uses the X11/NeWS operator **objectdump** and the **objectdiff** command. Each time **objectwatcher** is run, the current snapshot of the server obtained from **objectdump** is left in the file **/tmp/objects.latest** for future comparisons.

The first time **objectwatcher** is run, there is usually no **objects.latest** file in **/tmp**. This results in no output. If an **objects.latest** file exists in **/tmp** from a run of **objectwatcher** on a previous server, the output should be ignored as it is comparing different servers.

The output from **objectwatcher** is in the following format:

```
nnnnn bytes for mmm object_type objects
```

The output is sorted from the largest memory size allocated at the top to the smallest size allocated or the largest size freed at the bottom. There is a total-line at the end:

```
nnnnn bytes for mmm TOTAL objects
```

The *object_types* are the same as those used for the output from **objectdump**.

EXAMPLES

To find the number of objects allocated/dcallocated during some operation, set up the environment to test the operation in question. Execute **objectwatcher** to flush information about data objects allocated so far. Execute the operation in question. Execute **objectwatcher** again. The output indicates what allocations and deallocations of data objects occurred during processing of the operation:

```
example% objectwatcher
      % empty output or difference from last run
example% psh
executive
/Var 1 def
quit
example% objectwatcher
      % report of objects created or destroyed during the operation
```

SEE ALSO

objectdiff(1)

NAME

OpenWindows Demonstrations

SYNOPSIS

Demos menu item in the Programs submenu.

OVERVIEW

The **Demos** menu item on the root menu runs **hyperview**, a hypertext browser running a stack called **DemoNavigator**. This program allows you to browse around a hierarchy of X11 and NeWS demonstration programs. These programs are intended to demonstrate NeWS and X11 graphics and user interaction capabilities.

DESCRIPTION

The **Demos** are started by selecting the Demo item on the Programs pull-right menu on the root menu.

The NeWS Toolkit**PostScript Previewer****Color**

Peter

Tiger

Parrot

Chip

Black and White

Golfer

Rose

Shuttle

Nozzle

Porsche

Butterfly

Hawaii

Usamap

Worldmap

Multipage

Encapsulated PostScript

Overview

NeWS Rendering

Escher's Fish

World

SpaceShip

Lines

Spiral

Pie Chart

Wide Lines

X Logo

Rubber-band

Imaging

Text/Fonts

Scaled Text

Images

Magnifier

Animation

Technichron

Round Clock

Mona Eyes

Icosahedron

IcoScreenSaver
PolyScreenSaver
Flying Logos
Tetris
Wink
Colors
Color Names
Colormap
Color Wheel
Fader
Journaling
Calculator
X11 demos
xterm
XView PostScript
Ico
Solid Ico
Psycho
Maze
Muncher
Plaid
Puzzle
Worm
Xsol

The menu items are described below.

The News Toolkit: POSTSCRIPT Previewer

Color The following color demos:

- Peter** Peter Gabriel drawn by David Lavallee using Painter.
- Tiger** Bengal Tiger from Adobe Illustrator.
- Parrot** Colorful parrot from Adobe Illustrator.
- Chip** Custom Asic from a cad package.

Black and White

The following black and white demos:

- Golfer** The famous golfer from Adobe Illustrator.
- Rose** A vector drawing of a rose with a poem.
- Shuttle** AutoCAD cutaway drawing of a space shuttle.
- Nozzle** AutoCAD machanical drawing of a fire hose nozzle.
- Porsche**
Adobe Illustrator Porsche 911T.
- Butterfly**
A vector drawing of a butterfly.
- Hawaii** Map of Hawaii from SunDraw.
- Usamap**
Map of USA from Brian Reids netmap.
- Worldmap**
Map of the world from Brian Reids netmap.

Multipage

The following Multipage demos:

Encapsulated PostScript

The EPSF document from Adobe.

Overview

The NeWS Overview document done in Frame.

The NeWS Toolkit: NeWS Rendering**Escher's Fish**

Draws the famous *Square Limit* created by M. C. Escher. The demo is a 260-line recursive NeWS program that draws a large number of vectors. You can use the menu to vary the complexity of this drawing.

World Displays a geographic projection of the western hemisphere.

SpaceShip

A demonstration of NeWS' vector-drawing capabilities. The demo draws four spaceships inside its window, composed of over 7,000 vectors.

Lines Creates a window with a line pattern inside of it. You can alter the number of lines drawn from the pop-up menu inside the window. On color screens, the line pattern is displayed in a rainbow of colors.

Spiral Draws a simple spiral pattern.

Pie Chart

Draws a business pie chart with slices of the pie filled with varying colors.

Wide Lines

A simple sketchpad that demonstrates NeWS' wide line drawing capabilities. Click left to move the current point and middle to draw a line, curve or arc to any point. The menu allows you to change all of the possible options in the POSTSCRIPT language graphics context which affect wide line drawing.

X Logo Draws an X logo based on that by Danny Chong.

Rubber-band

Demonstrates how responsive NeWS can be when interacting with you. Click and drag the Point button to drag out a vector, rectangle or cubic spline curve. After a point is placed, you may adjust it by selecting it with the middle button.

The NeWS Toolkit: Imaging**Text/Fonts**

Writes text inside a window in several styles. The right button brings up a pop-up menu from which you can select the font under the *Font* pull-right, the point size, the colors, and the text to be shown. The text shown can be either some sample text or a list of all characters in the chosen font.

Scaled Text

Demonstrates NeWS' ability to scale text to an arbitrary size using the fill smart outline fonts of the imbedded OpenFonts technology of X11/NeWS.

Images Combines all of the image demos from NeWS 1.1 into one demo. The Images submenu lets you choose which image to manipulate while the Modes submenu lets you choose the mode you wish to view the image in. Scaled Image fits the image into the frame, whatever size you make it. Pan Image renders the image in its native size and lets you pan it around by pressing and dragging the left mouse button. If you "lose" the image, pressing the middle button will bring it back. Bounce Image will "automatically" pan the image around, bouncing off of the window borders. Tiled Image will render the image as many times as it takes to fill the window frame. Rotated image will render the image at several rotations around the clock. Spin image will render the image in a

user defined square at an arbitrary orientation. Press the left mouse button and drag out any square and the image will repaint inside it. The Triangle, Doughnut and SunLogo Stencils all demonstrate NeWS' POSTSCRIPT language imaging model, where all operations including imaging can be clipped to an arbitrary "stencil". The brightness and contrast menus can be used to adjust these viewing parameters using the POSTSCRIPT language transfer function.

Magnifier

The magnifier expands the bits under the cursor location. The zoom level can be controlled by the menu. You can stop the snapshots of the cursor by pressing the middle button. If you press left button in the magnifier window when it is stopped it will restart.

The NeWS Toolkit: Animation

Technichron

Technichron displays the time of day by showing how the light is falling on the earth at the current time. There are several methods of "Time Warp" available from the menu. These were originally intended for debugging purposes, but have been retained for their educational value.

Round Clock

A clock written entirely in the POSTSCRIPT language. It uses unorthodox methods of getting the time of day from the system, and overrides almost every possible method in the frame and canvas classes in the NeWS toolkit.

Mona Eyes

Represents half of an all-nighter and the lighter side of NeWS programming. The code is mostly stolen from NeWS 1.1's Eyecosahedron, the ideas are stolen from Monty Python's Flying Circus.

Icosahedron

Displays a bouncing 20 sided regular solid with the hidden lines removed. The menu switches between rendering directly to the framebuffer and buffering the rendering through an image. On machines with fast graphics hardware the former will be faster.

IcoScreenSaver

This is the same demo as above, but it covers the whole screen and goes away with a click of the mouse button.

PolyScreenSaver

Another screen saver, which is modeled after the Mesa program "Poly".

Flying Logos

An observation of the one-to-one-ness of NeWS and the Sun Logo.

Tetris

A straightforward port of the popular Game.

Wink

Displays a pair of eyes in the middle of the screen, one of which winks at you.

The NeWS Toolkit: Colors

Color Names

Shows you the correspondence between color names in the color dictionary as implemented by NeWS/colors.ps and their colors on the screen. This program uses scrollbars to access all the colors.

Colormap

Displays the installed colormap. It also installs its own Hue ramp on Enter and uninstalls it on Exit.

Color Wheel

Draws a wheel of colors inside a window. You can use the menu to switch between gray and color, and to vary the number of shades, the saturation, and the intensity of the colors displayed.

Fader

Uses colormap animation to fade between two different strings with an interesting visual effect. The menu is used to change the fading rate and to pick a different set of strings to fade.

Journaling

A demonstration of the Record and Play features of the NeWS event distribution system.

Calculator

The simple four-function calculator in Reverse Polish Notation style.

X11 demos

xterm The standard X11 terminal emulator.

XView PostScript

X11 access to PostScript revealed.

Ico The original Icosahedron.

Solid Ico

The same as Ico, but uses colormap double buffering.

Psycho Another Ico demo, which uses multiple windows.

Maze A non-interactive maze generator and solver. The program will generate and draw a maze in the window. It will then solve the maze using a backtracking algorithm, showing all of its attempts along the way.

Muncher

Tests graphics performance.

Plaid Another graphics performance test.

Puzzle A simulation of the 15 tiles in a 4x4 grid game.

Worm Slithering worms and a rotating colormap.

Xsol Solitaire simulator.

FILES

\$XNEWSHOME/etc/NeWS/hyperdecks/xnewsdemo.hv

the hyperview(1) deck which has the hypertext data for the DemoNavigator.

\$XNEWSHOME/demo/* All of the demo programs run by the DemoNavigator.

SEE ALSO

psh(1), pageview(1), hyperview(1)

NeWS Programmer's Guide

PostScript Language Reference Manual, Adobe Systems Inc., Addison-Wesley

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc.

NAME

pageview – POSTSCRIPT language previewer for NeWS

SYNOPSIS

```
pageview [ -displayNEWSSERVER ] [ -geometryWxH+X+Y ] [ -mono ] [ -paperwidth ] [ -paperheight ] [ -dpidots/inch ] [ -pagepage ] [ -dirdirectory ] [ -left|right|upsidedown ] [ -verbose ] [
psfile | - ]
```

DESCRIPTION

pageview is an interactive POSTSCRIPT previewer. Unlike its predecessor psview, pageview does not attempt to fit the whole page into a given window. pageview renders a document, a page at a time, onto an offscreen bitmap which may be of arbitrary size, resolution and orientation. The user can then adjust the viewing window's size to see as much of the page as desired. The mouse buttons are used to position the page under the window in two different modes.

The left button moves the page in "relative mode". This allows you to move the page in a physically intuitive way. You press the left button on the page and while you drag the mouse around, the spot on the page that was under the mouse cursor when you pressed the button remains stationary relative to the cursor.

The middle button moves the page in "absolute mode". This allows you to easily get to the edges of the document, especially when the DPI is large and/or the window is small. When you press the middle button on a point in the window, the page is adjusted so that the same relative point on the page is under the mouse. For example, if you press the middle button at the top right corner of the window, you will see the top right corner of the page. A little experimentation with a page at 300 dpi and you will find this mode indispensable.

There are several buttons across the top of the pageview window which are described below:

Page:

In multipage documents this allows you to move to the Next, Previous, First and Last pages. In single page documents this menu is disabled. To go to an arbitrary page use the slider at the bottom of the window, or type in the page number at the "Page: " prompt to the left of the slider.

DPI: This lets you change the "resolution" of the retained bitmap which the page is being rendered onto. 36 dpi will make a US Letter sized page be 306x396 pixels, where 300 dpi would be 2490x3300 pixels. This has the effect of making 36 dpi images appear smaller and 300 dpi pages appear larger due to the static resolution of the display. pageview starts out at 80 dpi, unless you have the environment variable \$DPI set to some other default, or you use the -dpi command line argument.

Size: This lets you change the size of the retained bitmap which the page is rendered onto. USLetter is 8.5x11, Legal is 8.5x14, and Envelope is 8.5x4.5. These values can be set to custom values by the -height and -width command line arguments.

Rotation:

This menu lets you choose which way to rotate the paper in 90 degree increments. This is useful for viewing slides which are commonly rendered in "Landscape left" orientation.

Print:

There are three options to printing pages from pageview. You can print the current page or the whole document on a II's laser printer. This simply sends the IIs for the current page to lpr(1), so you can use the \$PRINTER environment variable to set the printer to use. You can also dump the retained bitmap of the current page to a rasterfile(5) named /tmp/{ documentname }.{ page }.ras, where documentname and page are replaced by the appropriate values. This rasterfile can be edited by Sun-
Paint, or printed on any other device capable of rendering bitmaps.

OPTIONS

-display*NEWSERVER*

sets the NeWS server to connect to, this defaults to the value of the \$NEWSERVER environment variable. See *newserverstr*(1) to find out how to set this variable.

-geometry*WxH+X+Y*

sets the location and size of the outer frame of pageview. The Width and Height are in pixels and the X and Y specify the lower left corner. For example, "pageview -display 200x400+100+100" will start a pageview, 200 wide and 400 tall with the lower left corner at 100,100.

-mono

is used to force pageview to use a monochrome retained canvas on color systems. This saves memory and is faster on some framebuffer.

-wpaperwidth

sets the width of the "paper" to *paperwidth* inches, the default is 8.5.

-hpaperheight

sets the height of the "paper" to *paperheight* inches, the default is 11.

-dpi*dots/inch*

sets the "dpi" of the "paper" to *dots/inch*. The environment variable \$DPI is used if this option is not present, and the default is 80 if this variable is not in the environment. Caution must be used in setting this argument as well as the paper size args above, so you do not exhaust memory resources. For example a USLetter sized page previewed at 300 dpi, takes up 300*8.5/8*300*11 or a little over a Megabyte. The same page at 1500 dpi takes over 26 Megabytes.

-dir*directory*

Sets the current working directory to *directory* so that you can type filenames at the "Load File: " prompt relative to *directory*

-left|right|upside*down*

Sets the rotation of the page.

-verbose

Prints lots of debugging information (not useful to the user)

If *psfile* is specified, the POSTSCRIPT code is taken from that file.

If no argument is given, or if a '-' is given as the argument, pageview reads the POSTSCRIPT program from standard input.

SEE ALSO

psh(1), *newserverstr*(1), *lpr*(1)

POSTSCRIPT Language Reference Manual, Adobe Systems Inc., Addison-Wesley

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc

NAME

pam – remove sticky window from NeWS display

SYNOPSIS

pam

DESCRIPTION

pam is a **psh(1)** script that causes a specified window to be unmapped from the display. The command prompts the user to click a mouse button with the mouse positioned over the "stuck" window by drawing a lightning bolt near the cursor. If the window that is clicked on happens to be a frame for a tNt-based application, the window will be destroyed and the memory it occupies may be reclaimed depending on the situation that caused it to become stuck in the first place. This destruction is in addition to the standard unmaping action.

This tool is useful when an application has exited and left a window displayed that is not usable (due to an internal error). Using this command does not deallocate any memory or fix the problem, but it will unmap the window so it will no longer be visible.

SEE ALSO

psh(1)

NeWS Programmer's Guide

BUGS

For users to be able to type in **pam** to operate this command, **psh(1)** must be installed in **/usr/NeWS/bin/psh**. Otherwise, users must type **psh pam**.

NAME

psh – NeWS POSTSCRIPT shell

SYNOPSIS

psh [*files*]

DESCRIPTION

If a *files* argument is specified, **psh** opens a connection to the server and transmits the specified files to the server. If no *files* argument is specified, or if '-' is specified, **psh** opens a connection to the server and transmits stdin to the server. Any output from NeWS is copied to stdout. The files should be POSTSCRIPT programs for the NeWS server to execute.

A common use for **psh** is in creating applications written entirely in the POSTSCRIPT language. First, type your POSTSCRIPT program into a file. Then, add this as its first line:

```
#!/usr/NeWS/bin/psh
```

If you now make the file executable (with **chmod**), you can invoke it by name from the shell, and UNIX will use **/usr/NeWS/bin/psh** to execute it. **psh** will in turn send your program to the X11/NeWS server.

SEE ALSO

sh(1)

NeWS Programmer's Guide

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc.

BUGS

If **#!/usr/NeWS/bin/psh** is the first line, the script or file is implementation or installation dependent.

PSH(1)

USER COMMANDS

PSH(1)

NAME

psindent – format POSTSCRIPT language or NeWS source

SYNOPSIS

```
psindent [ [-ba|+ba] [-bb|+bb] [-bs|+bs] [-cba|+cba] [-cbb|+cbb] [-cea|+cea]
[-ceb|+ceb] [-da|+da] [-db|+db] [-dba|+dba] [-dbb|+dbb] [-dea|+dea] [-deb|+deb]
[-ea|+ea] [-eb|+eb] [-gra|+gra] [-grb|+grb] [-gsa|+gsa] [-gsb|+gsb] [-ia|+ia]
[-ib|+ib] [-ib|+ib] [-lba|+lba] [-lbb|+lbb] [-lca|+lca] [-lcb|+lcb] [-rba|+rba]
[-rbb|+rbb] [-rca|+rca] [-rcb|+rcb] ] [input-file ... ]
```

DESCRIPTION

psindent is a POSTSCRIPT language formatter that also understands NeWS elements. It reformats the POSTSCRIPT language program in one or more *input-files*, according to the switches. The switches that can be specified are described below. They may appear before or after the filenames. If only the *input-file* is specified, the formatting is done *in place*, that is, the formatted file is written back into *input-file*, after a backup copy named *input-file.BAK* has been written in the current directory. If no *input-file* is given, psindent acts as a filter: the standard input is formatted and the results written to the standard output. This is most useful from within editors.

In addition to being specified on the command line, switches may be given in the user's own profile of defaults. When psindent is run, it checks for the file .psindent in the user's login directory. If that file exists, it is read to set the defaults of psindent. However, switches on the command line always override profile switches. The switches should be separated by SPACE, TAB, or NEWLINE characters. Finally, switches may also be indicated by embedding them in the source code itself, at the beginning of a comment line.

Lines beginning with a # or a % are passed through unmodified. Lines beginning with the keyword cdef are assumed to be CPS (C to POSTSCRIPT) calls under NeWS. This causes the lines until either the next cdef or the end of the file to be indented once.

OPTIONS

The options listed below control the formatting style imposed by psindent. A + sets the option, and a - deselects it.

-ba,+ba	If ba is specified, a NEWLINE is forced after every begin. Default: +ba
-bb,+bb	If bb is specified, a NEWLINE is forced before every begin. Default: +bb.
-bs,+bs	Backslash inner strings, such as (\()). Default: +bs.
-cba,+cba	If cba is specified, a NEWLINE is forced after every classbegin. Default: +cba.
-cbb,+cbb	If cbb is specified, a NEWLINE is forced before every classbegin. Default: +cbb.
-cea,+cea	If cea is specified, a NEWLINE is forced after every classend. Default: +cea.
-ceb,+ceb	If ceb is specified, a NEWLINE is forced before every classend. Default: +ceb.
-da,+da	If da is specified, a NEWLINE is forced after every def. Default: +da.
-db,+db	If db is specified, a NEWLINE is forced before every def. Default: -db.
-dba,+dba	If dba is specified, a NEWLINE is forced after every dictbegin. Default: +dba.
-dbb,+dbb	If dbb is specified, a NEWLINE is forced before every dictbegin. Default: +dbb.
-dea,+dea	If dea is specified, a NEWLINE is forced after every dictend. Default: +dea.
-deb,+deb	If deb is specified, a NEWLINE is forced before every dictend. Default: +deb.
-ea,+ea	If ea is specified, a NEWLINE is forced after every end. Default: +ea.
-eb,+eb	If eb is specified, a NEWLINE is forced before every end. Default: +eb.
-gra,+gra	If gra is specified, a NEWLINE is forced after every grestore. Default: -gra.
-grb,+grb	If grb is specified, a NEWLINE is forced before every grestore. Default: -grb.

-gsa,+gsa	If gsa is specified, a NEWLINE is forced after every gsave . Default: -gsa .
-gsb,+gsb	If gsb is specified, a NEWLINE is forced before every gsave . Default: -gsb .
-ia,+ia	If ia is specified, a NEWLINE is forced after every if or ifelse . Default: +ia .
-ib,+ib	If ib is specified, a NEWLINE is forced before every if or ifelse . Default: -ib .
-lba,+lba	If lba is specified, a NEWLINE is forced after every [. Default: -lba .
-lbb,+lbb	If lbb is specified, a NEWLINE is forced before every [. Default: -lbb .
-lca,+lca	If lca is specified, a NEWLINE is forced after every { . Default: +lca .
-lcb,+lcb	If lcb is specified, a NEWLINE is forced before every { . Default: -lcb .
-rba,+rba	If rba is specified, a NEWLINE is forced after every] . Default: -rba .
-rbb,+rbb	If rbb is specified, a NEWLINE is forced before every] . Default: -rbb .
-rca,+rca	If rca is specified, a NEWLINE is forced after every } . Default: -rca .
-rcb,+rcb	If rcb is specified, a NEWLINE is forced before every } . Default: +rcb .

EXAMPLES

The following illustrate the three methods for setting option switches.

On the command line:

```
psindent -bs +cea -dba file_name.ps
```

In the file `~/psindent`:

```
-bs +cea -dba
```

Within a source file *file.ps*:

```
<beginning-of-line>% %= -bs +cea -dba
```

From within `vi(1)` the editing buffer may be formatted by typing:

```
:%!psindent
```

FILES

```
~/psindent          profile file
```

SEE ALSO

`pstags(1)`

NOTES

`psindent(1)` and `pstags(1)` are the same program.

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc.

BUGS

`psindent` switches embedded in the source are only recognized at the beginning of a comment line. If a profile file exists in the **current** directory, it should be read, and its settings should have precedence over those in the profile file in the login directory, if one exists.

AUTHOR

Josh Siegel (`siegel@hc.dspo.gov`)

NAME

psio – NeWS buffered input/output package

SYNOPSIS

```
#include <NeWS/psio.h>
```

```
psio_assoc(pinstream, postream)
```

```
PSFILE *pinstream;
```

```
PSFILE *postream;
```

```
psio_availinputbytes(pstream)
```

```
PSFILE *pstream;
```

```
psio_availoutputbytes(pstream)
```

```
PSFILE *pstream;
```

```
psio_bytesoutput(pstream)
```

```
PSFILE *pstream
```

```
void psio_clearerr(pstream)
```

```
PSFILE *pstream;
```

```
psio_clearnonblock(pstream)
```

```
PSFILE *pstream;
```

```
int psio_close(pstream)
```

```
PSFILE *pstream;
```

```
psio_eof(pstream)
```

```
PSFILE *pstream;
```

```
psio_error(pstream)
```

```
PSFILE *pstream;
```

```
PSFILE *psio_fdopen(fd, mode)
```

```
int fd;
```

```
char *mode;
```

```
int psio_flush(pstream)
```

```
PSFILE *pstream;
```

```
int psio_fileno(pstream)
```

```
PSFILE *pstream;
```

```
psio_fprintf(pstream, format[,arg]...)
```

```
PSFILE *pstream;
```

```
char *format;
```

```
PSFILE *psio_getassoc(pinstream)
```

```
PSFILE *pinstream;
```

```
int psio_getc(pstream)
```

```
PSFILE *pstream;
```

```
PSFILE *psio_open(fname, mode)
char *fname;
char *mode;

psio_pgetc(pstream, dest, pausecode)
PSFILE *pstream;
int dest;
int pausecode;

psio_pgetc_nb(pstream, dest, pausecode)
PSFILE *pstream;
int dest;
int pausecode;

psio_pputc(c, pstream, pausecode)
char c;
PSFILE *pstream;
int pausecode;

psio_printf(format [,arg]...)
char *format;

psio_putc(ch, pstream)
char ch;
PSFILE *pstream;

int psio_read(dest, size, num, pstream)
unsigned char *dest;
int size, num;
PSFILE *pstream;

psio_setnonblock(pstream)
PSFILE *pstream;

PSFILE *psio_sopen(*string, length, *mode)
char *string;
int length;
char *mode;

psio_ungetc(ch, pstream)
char ch;
PSFILE *pstream;

psio_write(source, size, num, pstream)
char *source;
int size, num;
PSFILE *pstream;
```

DESCRIPTION

The functions described here constitute a user-level I/O buffering scheme for use when communicating with NeWS. This package is based on the standard I/O package that comes with Unix. The functions in this package are used in the same way as the similarly named functions in Standard I/O.

The macros `psio_getc` and `psio_putc` read and write single characters quickly. The higher level routines `psio_read`, `psio_printf`, `psio_fprintf`, `psio_write` all use or act as if they use `psio_getc` and `psio_putc`; they can be freely intermixed.

A file with associated buffering is called a *stream* and is declared to be a pointer to a defined type `PSFILE`. `psio_open` creates certain descriptive data for a stream and returns a pointer to designate the stream in all further transactions. Normally, there are three open streams with constant pointers declared in the `psio.h` include file and associated with the standard open files:

<code>psio_stdin</code>	standard input file
<code>psio_stdout</code>	standard output file
<code>psio_stderr</code>	standard error file

Any module that uses this package must include the header file of pertinent macro definitions, as follows:

```
#include "psio.h"
```

The functions and constants mentioned here are declared in that header file and need no further declaration. The constants and the following 'functions' are implemented as macros (redeclaration of these names is perilous): `psio_getc`, `psio_putc`, `psio_eof`, `psio_error`, `psio_fileno`, `psio_clearerr`, `psio_pgetc`, `psio_pgetc_nb`, `psio_pputc`, `psio_setnonblock`, `psio_clearnonblock`, `psio_assoc`, `psio_getassoc`, `psio_availinputbytes`, `psio_availoutputbytes`, and `psio_bytesoutput`.

The `psio` package contains enhancements over the `stdio` package that the X11/NeWS server and CPS clients need to utilize. These features include:

The ability to open a `psio` stream on an in-memory string using the `psio_sopen` function.

Support for non-blocking I/O. If you set the file descriptors of the `psio` streams for non-blocking I/O, the actions that do not expect to see non-blocking behavior, such as `psio_getc`, `psio_putc` and the functions that use them, will still behave correctly, blocking on the file descriptor until the I/O completes. In order to access the streams in a non-blocking mode, there are macros to perform non-blocking character reads and writes: `psio_pgetc`, `psio_pgetc_nb`, and `psio_pputc`. The `psio` package also performs non-blocking I/O operations on the file descriptor for you, without your having to use system dependent functions to set up the file descriptors for non-blocking I/O, as specified by the `psio_setnonblock` macro.

Support for buffer *look-aheads*. The CPS package sometimes needs to skip over some of the data in the `psio` buffer and read data that is not at the front of the buffer. The `psio` package cooperates with the CPS package in this respect.

Support for growable buffers. If a stream is set for non-blocking output and overfills its buffer, or if a buffer *look-ahead* cannot find the data it needs in the existing buffer, `psio` automatically makes the buffer grow in order to allow the writes to continue without blocking, and to allow the *look-ahead* to succeed.

Support for linked input/output streams. Using the `psio_assoc` macro, you can associate an output stream with an input stream so that the output stream is flushed whenever the `psio` package needs to block on the input stream.

Macros to query the number of bytes available for reading, or the number of bytes available for output.

LIST OF FUNCTIONS:

<i>Name</i>	<i>Description</i>
psio_assoc	Associate the specified output stream with the specified input stream. The output stream will be flushed when psio needs to block on the input stream. Implemented as a macro.
psio_availinputbytes	Return the number of bytes currently in the buffer waiting to be read. Implemented as a macro.
psio_availoutputbytes	Return the number of bytes that can be written to the stream before the buffer is filled. Implemented as a macro.
psio_bytesoutput	Return the number of bytes currently in the buffer that have been written and are waiting to be flushed. Implemented as a macro.
psio_clearerr	Clear the error and end-of-file flags for the specified stream. Implemented as a macro.
psio_clearnonblock	Turn off the psio automatic non-blocking I/O feature for the specified stream. Implemented as a macro.
psio_close	Close the file associated with the stream and free the associated memory.
psio_eof	Check the stream for a previously detected end-of-file status. Implemented as a macro.
psio_error	Check the stream for a previously detected error. Implemented as a macro.
psio_fdopen	Open a stream and associate it with the specified file descriptor.
psio_flush	Write any pending output for the specified output stream.
psio_fileno	Return the file descriptor associated with the specified stream. Implemented as a macro.
psio_fprintf	Place output onto the named output stream according to standard printf specifications.
psio_getassoc	Return the output stream associated with the specified input stream. Implemented as a macro.
psio_getc	Get a character or EOF from the specified input stream. Implemented as a macro.
psio_getc	Like psio_getc , except that the result of the getc is assigned to <i>dest</i> , and if the operation needs to block, pausecode is executed first. pausecode is executed every time the buffer is emptied. An attempt to refill the buffer is made before the pausecode is executed, so that the pausecode can determine if more input is available. Implemented as a macro.
psio_pgetc_nb	Like psio_pgetc , except that it tries to fill the buffer before pausing. This is usually desirable for the first read on the stream after it has just been created or previously blocked (for example, in a psio_pgetc call). Implemented as a macro.
psio_open	Open the named file with the specified mode.
psio_printf	Place output onto the stream psio_stdout according to standard printf specifications.
psio_putc	Write a character to the specified output stream. Implemented as a macro.
psio_pputc	Like psio_putc , except that if the operation needs to block, pausecode is executed first. pausecode is executed every time the buffer is flushed.
psio_read	Read <i>num</i> blocks of <i>size</i> bytes from the specified input stream into the buffer <i>dest</i> .
psio_setnonblock	Indicate that non-blocking I/O is to be performed on the specified stream. The psio package will set the corresponding file descriptor for non-blocking I/O during its reads and writes to prevent blocking. Note that certain functions (such as psio_getc) will

block anyway, and that it is faster (though less elegant and portable) if you yourself set the file descriptor once for non-blocking I/O, rather than using this feature to have `psio` set and reset the file descriptor every time a read or write is performed. Implemented as a macro.

- psio_sopen** Open the in-memory string of length *length* for reading or writing depending on *mode*. If *string* is `NULL`, then `psio` will allocate a buffer for the result (for writing only) of the specified length; this buffer will be grown if writes overrun the specified length.
- psio_ungetc** Push the character *ch* back into the specified input stream.
- psio_write** Write *num* blocks of *size* bytes to the specified output stream from the buffer *source*.

DIAGNOSTICS

The value `EOF` is returned uniformly to indicate that a `PSFILE` pointer has not been initialized with `psio_open`, that input has been attempted on an output stream, that output has been attempted on an input stream, or that a `PSFILE` pointer designates corrupt or otherwise unintelligible `PSFILE` data. An integer constant `(-1)` is returned upon end-of-file or error by most integer functions that deal with streams. `psio_open` returns a pointer to the `psio` stream or `NULL` (`0`) if there is an error.

SEE ALSO

`intro(3S)`, `fclose(3S)`, `ferror(3S)`, `fopen(3S)`, `fread(3S)`, `getc(3S)`, `printf(3S)`, `putc(3S)`, `ungetc(3S)`.

NAME

psman – display reference manual pages; find reference pages by keyword

SYNOPSIS

psman [**-M** *path*] [*section*] *title*
psman [**-M** *path*] **-k** *keyword* ...

DESCRIPTION

psman displays information from the reference manuals. It can display complete manual pages that you select by *title*. It can display one-line summaries selected by **-k** *keyword*.

When **-k** is not specified, **psman** formats a specified manual page by *title*. A *section*, when given, applies to the *title* that follows it on the command line. **psman** looks in the indicated section of the manual for the *title*. *section* should be a digit. If *section* is omitted, **psman** searches all reference sections (giving preference to commands over functions) and prints the first manual page it finds. If no manual page is located, **psman** prints an error message.

The following line instructs **psman** to look in section 8 of the reference manual for the **ypwhich(8)** manual page:

```
example% psman 8 ypwhich
```

If the server is not available **psman** formats for a teletype and pipes its output through **more(1)**. Otherwise, **psman** formats for a POSTSCRIPT printer and pipes its output through **pageview(1)**. To see the manual page for **pageview**, use:

```
example% psman pageview
```

OPTIONS

-M *path*

Change the search path for manual pages. *path* is a colon-separated list of directories that contain manual page directory subtrees. For example, **/usr/share/man:/home/openwin/usr/share/man** makes **psman** search in the standard OpenWindows location. When used with the **-k** option, the **-M** option must appear first. Each directory in the *path* is assumed to contain subdirectories of the form **man[1-8l-p]**.

-k *keyword* ...

psman prints out one-line summaries from the **whatis** database (table of contents) that contain any of the given *keywords*.

ENVIRONMENT

MANPATH If set, its value overrides **/usr/man:\$OPENWINHOME/man** as the default search path.

TROFF If set, its value overrides **ditroff -t -man** as the default command to convert troff to ditroff.

TCAT If set, its value overrides **psdit** as the default command to convert ditroff to POSTSCRIPT.

SEE ALSO

pageview(1), **cat(1V)**, **col(1V)**, **eqn(1)**, **more(1)**, **nroff(1)**, **tbl(1)**, **troff(1)**, **whatis(1)**, **man(7)**, **catman(8)**

NAME

psps – NeWS process lister

SYNOPSIS

psps

DESCRIPTION

psps is a **psh(1)** script that prints information for every lightweight process in the X11/NeWS server. The output from this command consists of the following eight columns of information for each process.

> The symbol **>** in the first column indicates that the process is the first process in its process group. If there are other processes in the process group, they will be listed after this first process, and they will each have a blank space in the first column.

ID The second column gives the ID number associated with the NeWS process.

State The third column gives the state of the NeWS process. (See the description of the process dictionary keys in the NeWS Programmer's Guide.)

Pri The fourth column gives the priority of the NeWS process. (See the description of the process dictionary keys in the NeWS Programmer's Guide.)

ESS The fifth column gives the size of the NeWS process' execution stack.

OSS The sixth column gives the size of the NeWS process' operand stack.

DSS The seventh column gives the size of the NeWS process' dictionary stack.

Name The eighth column gives the value of the process' ProcessName key. (See the description of the process dictionary keys in the NeWS Programmer's Guide.)

OPTIONS

None.

SEE ALSO

psh(1)

NeWS Programmer's Guide

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc.

PSPS(1)

USER COMMANDS

PSPS(1)

NAME

`pstags` – create a POSTSCRIPT language or NEWS tags file for use with `vi`

SYNOPSIS

`pstags [-f tagsfile] [files]`

DESCRIPTION

`pstags` makes a tags file for `ex(1)` from a POSTSCRIPT language or NEWS source file. See `ctags(1)` for more information. `pstags` tags lines with the string `cdef` and lines with the hint `pstag=<tagname>` embedded anywhere in a comment.

The default tagfile is `tags.ps`. Because this is not in the default tag search path, the command

```
set tags=tags tags.ps
```

is required in `vi(1)`, or in the user's `.exrc` file to initialize that path. Depending on the system, a backslash may be required to escape the space between `tags` and `tags.ps`.

OPTIONS

None.

EXAMPLES

Using `pstags` on a file containing the function definition

```
cdef function_name(int arg)
```

produces the tag `function_name`.

Inserting the comment:

```
 / name { % object => - pstag=<name>
```

in the source file yields the tag `name` for that line. Note that it is the string within the `<...>` that is used as the tag.

FILES

`tags.ps` output tags file

SEE ALSO

`ctags(1)`, `ex(1)`, `psindent(1)`, `vgrind(1)`, `vi(1)`

BUGS

Should be able to identify the tags without the hint in the comment.

AUTHOR

Josh Siegel (siegel@hc.dspo.gov)

PSTAGS(1)

USER COMMANDS

PSTAGS(1)

NAME

pstern — NeWS terminal emulator

SYNOPSIS

pstern [*options*] [*command*]

DESCRIPTION

pstern is a termcap-based terminal emulator program for NeWS. When invoked, it reads the */etc/termcap* entry for the terminal named by the *-t* option, or by the *TERM* environment variable, and arranges to emulate the behavior of that terminal. It forks an instance of *command* (or, by default, the program specified by the *SHELL* environment variable, or *cs*h if this is undefined), routing keyboard input to the program and displaying its output.

pstern scales its font to make the number of rows and columns specified in the */etc/termcap* entry for the terminal it is emulating fit the size of its window. It also responds to most of the particular escape sequences that *termcap* defines for that terminal.

OPTIONS

- C* Route */dev/console* messages to this window, if supported by the operating system.
- f* Bring up a reasonably-sized terminal in the lower-left corner of the screen (or in the location specified with the *-xy* option) instead of having the user define its size and location.
- w* Wait around after the *command* terminates.
- fl* *framelabel*
Use the specified string for the frame label.
- il* *iconlabel*
Use the specified string for the icon label. The icon label normally defaults to the name of the host on which pstern is running.
- li* *lines* Specifies the height of the window in characters.
- co* *columns*
Specifies the width of the window in characters.
- xy* *x y* Specifies the location of the lower left hand corner of the window (in screen pixel coordinates).
- bg* Causes pstern to place itself in the background by disassociating itself from the parent process and the controlling terminal. If pstern is invoked with *rsh*(1), this option will cause the *rsh* command to complete immediately, rather than hang around until pstern exits.
- ls* causes pstern to invoke the shell as a login shell. In addition, any specified *command* will be passed to the shell with a *-c* option, rather than being invoked directly, so that the shell can establish any environment variables that may be needed by the command. Further, if pstern is invoked via *rsh*(1), the host at the other end of the *rsh* socket will be used as the server, unless a *NEWSERVER* environment variable is present.
- pm* Specifies that a pstern should enable *page mode*. When page mode is enabled and a command produces more lines of output that can fit on the screen at once, pstern will stop scrolling, hide the cursor, and wait until the user types a character before resuming output. When pstern is blocked with a screenful of data, typing a carriage return or space will cause scrolling to proceed by one line or one screenful, respectively; any other character will cause the next screenful to appear and be passed through as normal input. This mode can also be enabled or disabled interactively, using the *Page Mode* menu item.
- t* Specifies the terminal type, which can be identified by the name of its *termcap* entry. For example:

```
example% pstern -t sun
example% pstern -t vt100
```

-fontsize size

Specifies the point size of the font to be used when `pstern` is being brought up in fixed size mode (see the `-f` option). The default size is 12 points.

SELECTION

Clicking the left mouse button over a character selects that character. Clicking it beyond the end of the line selects the newline at the end of that line. Clicking the middle mouse button over a character when a primary selection does not exist in that window selects that character. Clicking the middle mouse button over a character when a primary selection does exist in that window extends or shrinks the selection to that character. Pressing the left button and dragging the pointer over the text selects the text between the original press location and the current mouse location. Pressing the left button over previously select text and then dragging invokes the OPEN LOOK-style *drag and drop* selection mechanism.

The Copy key (L6) copies the *primary* selection to the *shelf*. The Paste key (L8) copies the contents of the *shelf* to the *insertion point*.

If you make a selection while holding down the Copy key, the selection will be a secondary selection. Subsequently letting go of the Copy key copies the *secondary* selection to the insertion point in the window that had the keyboard focus when the selection was begun.

Making a selection while holding down the Paste key also makes a secondary selection. It pastes the *primary* selection to the location of the secondary selection and deselects the secondary selection.

Copy and Paste of both primary and secondary selections work across separate invocations of `pstern`.

MENU ITEMS

`pstern` adds two items to the top of the standard menu associated with the right hand mouse button. These items permit the page mode and automatic margin modes to be turned on and off. Menu items change according to the state of each mode. For example, if page mode is enabled, the menu item will indicate **Page Mode Off**. The Automatic Margin entry controls the automatic wrapping of text characters when the text cursor hits the right terminal margin. When it is on, the text cursor and characters are automatically wrapped to the next line. When it is off, the text cursor remains at the terminal's right margin and characters overstrike one another in the last column.

FONTS

The `pstern` commands uses a `NeWS` class variable to decide which font to use. To select a font other than the default (which is `Courier`), place the following code in your `.startup.ps` file:

```
UserProfile begin
  /ClassPsTermCanvas {
    begin
      /TextFamily myfont store
    currentdict end
  } def
end
```

Here, `myfont` can be `/Courier`, `/fixed`, or `/LucidaSansTypewriter`. Any font that the server can access can be used; however, only fixed-width fonts work correctly.

The size of the font used is based on the size of the window. If the `-f` fixed option is used, `pstern` starts out at an appropriate size to use a 12 point version of the specified font family, unless the `-fontsize` option is used to specify a different point size. Even if `pstern` is started in fixed size mode, resizing it causes it to select a new font size to fit the new-sized window with the original number of rows and columns.

FILES

`/etc/termcap` to find the terminal description.

SEE ALSO

NeWS Programmer's Guide

BUGS

Emulating some terminal types works better than others, largely because there are incomplete */etc/termcap* entries for them.

A large number of *termcap* fields have yet to be implemented.

Page Mode gets easily confused.

Resizing *pstern* to a size smaller than the fixed startup size for bitmap fonts such as *fixed* causes display-garbage, since X11/NEWS cannot scale these fonts.

PSTERM(1)

USER COMMANDS

PSTERM(1)

NAME

xnews — window system server

SYNOPSIS

xnews [*POSTSCRIPT-code*]

DESCRIPTION

The xnews command starts the X11/NEWS window system server. The server supports both the X11 protocol and the NEWS protocol. Clients connect to the server by opening the socket appropriate to the protocol they use. The X11 protocol is described in the *X Window System Protocol, Version 11*, from MIT. The C language binding to the X11 protocol is described in *Xlib - C Language X Interface, Version 11*, also from MIT.

The NEWS protocol is a superset of the POSTSCRIPT page description language, described in the *POSTSCRIPT Language Reference Manual* by Adobe Systems, Inc. NEWS extensions to the POSTSCRIPT language are described in *The NEWS Programmer's Guide*, from Sun. The C client interface is also described there.

OPTIONS

[*POSTSCRIPT-code*]

The server interprets the POSTSCRIPT language code given as an argument on the command line. If no code is specified on the command line, xnews executes the following:

```
(NeWS/init.ps) (r) file cvx exec &main
```

This code fragment sets up the X11/NEWS server for its normal use. When specifying this argument, putting single quotes around the *POSTSCRIPT-code* will protect it from premature interpretation by the shell.

ENVIRONMENT VARIABLES

The X11/NEWS server recognizes the following environment variables:

FRAMEBUFFER

If unset, the default is `/dev/fb`. When the server starts up, it opens the display device named by `FRAMEBUFFER`.

OPENWINHOME

Should be set to the directory in which the servers' directory hierarchy is installed. It is used to initialize the *root menu* to point to the correct copies of the programs it invokes and to determine which directories are searched for POSTSCRIPT initialization files, fonts, and other libraries and programs.

XNEWSHOME

Overrides `OPENWINHOME` for the server if it is installed in a non-standard place. Ordinarily it should be unset. In this case, the server will set it automatically from `OPENWINHOME`.

XVIEWHOME

Overrides `OPENWINHOME` for XView applications if they are installed in a nonstandard place. Ordinarily it should be unset. In this case, the server will set it automatically from `OPENWINHOME`.

FONTPATH

A list of directories, separated by semicolons, telling the server where to look for fonts. If unset, it defaults to `$OPENWINHOME/lib/fonts`.

LD_LIBRARY_PATH

This variable is used by the shared library linker to determine which directories to examine for shared libraries. It should normally be set to `$OPENWINHOME/lib:/usr/lib`.

NEWSOCKET

When starting the server, if `NEWSOCKET` is set, the server will listen on the socket named by `NEWSOCKET` for NEWS clients. If `NEWSOCKET` is unset, the server will first try to open the

restricted socket number 144. If that fails, it will try 2000. If that fails, it will keep incrementing the socket number by 1 until it finds one on which it succeeds.

The socket on which the server listens for NeWS clients may also be set in your `~/startup.ps` file with a line of the following form:

```
/NeWS_socket (%socket12001) def
```

The `NeWS_socket` POSTSCRIPT language variable overrides the `NEWSOCKET` environment variable.

NEWSSERVER

Before starting a NeWS application, you can set this environment variable to tell the application which server to connect to. If unset, the application will default to the local server, socket 144. If socket 144 fails, it will try socket 2000. (For information on how to set this environment variable, see the manual page for `newsserverstr`.)

DISPLAY

Before starting an X11 application, you can set this environment variable to tell the application which server to connect to. If unset, you must specify the server on the application's command-line with the `-display` option. It is of the following form:

```
hostname:display.screen
```

Here, *hostname* may be the name of a host, `unix`, or `localhost`. The *display* argument is normally 0. The *.screen* argument is optional.

NOSXSEL

If set, the server does not start up `sv_xv_sel_svc`, which is used for cut and paste between SunView and XView programs.

X11ONLY

Reduces memory consumption if no NeWS clients will be run. Causes the server to initialize without starting the POSTSCRIPT window manager, `pswm`, or loading the NeWS toolkit, which `pswm` depends on. Additionally, the NeWS connection listener is not started. The documentation for the features that `X11ONLY` turns off does not necessarily describe the effects of `X11ONLY`.

NEWSONLY

Reduces memory consumption if no X11 clients will be run. Causes the server to initialize without constructing data structures required by the X11 interpreter or making X11 specific NeWS operators available. It also avoids starting the POSTSCRIPT window manager, `pswm`, although it still loads the NeWS toolkit, and avoids starting the X11 connection listener. Additionally, it omits X11 programs from the rootmenu. The documentation for the features that `NEWSONLY` turns off does not necessarily describe the effects of `NEWSONLY`.

NOPSWM

Causes the server to initialize without starting the POSTSCRIPT window manager, `pswm`. Unlike `X11ONLY`, the objective is not to save memory consumption but rather to make it easier to run a different X11 window manager.

USAGE

Getting Started

To start the X11/NeWS server, check where the server's directory hierarchy is installed. The recommended place is `/home/openwin`. Assuming it is installed on `/home/openwin`, set up your environment in the following way:

```
setenv OPENWINHOME /home/openwin
setenv LD_LIBRARY_PATH "$OPENWINHOME/lib:/usr/lib"
```

In your `.login` or `.cshrc` file, add `$OPENWINHOME/bin` and `$OPENWINHOME/bin/xview` to your `PATH` ahead of `/usr/bin`. If you have been running `NEWS`, you may have already set the `FONTPATH` variable. If so, unset it as follows:

```
unsetenv FONTPATH
```

If you have been running `NEWS 1.1`, you may have already set the `FRAMEBUFFER` variable. If so, unset it as follows:

```
unsetenv FRAMEBUFFER
```

Multiple Framebuffers

If you have two monitors, or if you have a `cg4` that you prefer to use as two framebuffers, you may wish either to run the `X11/NEWS` server on both or to run the server on one and `SunView` on the other.

To run `SunView` on one monitor and the `X11/NEWS` server on the other, first start `sunview` with the `-d` option. From `SunView`, set `FRAMEBUFFER` to the name of the other framebuffer. Then enter the following:

```
unsetenv WINDOW_PARENT
```

To start the `X11/NEWS` server, you can enter the following:

```
xnews &; sleep 12; adjacentscreens <sunview-dev> -r $FRAMEBUFFER
```

The `sunview-dev` argument is the name of the `/dev` device file that you started `SunView` on.

Alternately, you can add the following line to your `~/startup.ps` file:

```
(adjacentscreens <sunview-dev> -r $FRAMEBUFFER) runprogram
```

When you have added the line, you can start the server simply by typing `xnews &`.

You can then move the mouse back and forth between the two monitors, with `SunView` on the left and `X11/NEWS` on the right. See `adjacentscreens` for changing the configuration. If you want to start `SunView` applications from the terminal emulator in which you unset the `WINDOW_PARENT` environment variable, you must reset it to `/dev/win0`.

To run the `X11/NEWS` server on both monitors, you must start up two copies of the server, one for each framebuffer: this is necessary because support for two framebuffers from one server is not available yet. The following shell script takes two arguments (the framebuffer on the left and the framebuffer on the right), starts two copies of the server, and allows the mouse to move between them.

```
#!/bin/sh
export FRAMEBUFFER; FRAMEBUFFER=$1; xnews &
sleep 30
export FRAMEBUFFER; FRAMEBUFFER=$2; xnews &
sleep 12
adjacentscreens $1 -r $2
```

Alternately, if you always start the first server on `dev1` and the second server on `dev2`, add the following to your `.startup.ps` file:

```
(FRAMEBUFFER) () ?getenv
(dev2) eq {
  (adjacentscreens dev1 -r dev2) runprogram
```

```
} if
```

To treat a cg4 as two framebuffer, become superuser and enter the following:

```
cd /dev
MAKEDEV /dev/bwtwo0
ln -s fb cgfour0_clr
```

When the X11/NEWS server is started on /dev/fb on a cg4, it displays in the color plane group and displays the cursor in the monochrome plane. If you switch either to another X11/NEWS server or to SunView running in the monochrome plane using adjacentscreens, the cursor leaves a little blotch in the other display. To avoid the little blotch, use cgfour0_clr instead of fb.

So, to run SunView in monochrome and X11/NEWS in color, start sunview -d /dev/bwtwo0, and start X11/NEWS with FRAMEBUFFER set to /dev/cgfour0_clr. To run SunView in color and X11/NEWS in monochrome, start SunView on /dev/fb, and start the X11/NEWS server with FRAMEBUFFER set to /dev/bwtwo0. To run two copies of the X11/NEWS server, use the above script with the arguments /dev/bwtwo0 and /dev/cgfour0_clr.

After Startup

Immediately after starting the server, the console is set to be the workstation screen. This means that messages to the console will disrupt the entire window system display. To avoid disruption, a console window should be started as soon as possible after system initialization.

A console window is available from the root pop-up menu, in the Utilities pull-right menu.

Root Menu

The Root Menu user interface follows the *OPEN LOOK Graphical User Interface* specification.

To pop up the root menu, position the cursor anywhere in the root background and press the Menu button on the mouse. By default, the Menu button is the right button. If you click the button (that is, release it immediately after pressing it), the menu will stay up. You can make selections in the menu by clicking the Menu button over the items in the menu, or you may dismiss the menu by clicking the Menu button in the root background.

Menu items with an arrow on the right denote pull-right menus. Clicking the Menu button in these will pop up another level of menu.

You can also use mouse-drag to operate menus. If you press the Menu button down and hold it down, the menu will appear. As you move the mouse through the menu, the highlight will track the mouse. If you move the mouse to the right side of a pull-right item, the next level of menu will appear. To select a menu item in this mode, simply let go of the Menu button while the cursor is over your selection. To dismiss the menu, let go of the Menu button while the cursor is not over the menu.

Window Management

The window management user interface follows the *OPEN LOOK Graphical User Interface* specification.

When you start an application, it may prompt you with a *crosshair cursor* to drag out a rectangle the size of the new window. When it appears, its window comes up surrounded by a *window frame*. In the corners of the window frame are symbols called *resize corners*. The Select button, which by default is the left mouse button, may be used to drag a resize corner to the desired size. The symbol in the upper left hand corner may be clicked with the Select button to close the window, by default.

Pressing the Menu button anywhere in the window frame pops up the *window menu*. The window menu is operated the same way as the root menu.

FILES

If xnews is unable to open a file whose name does not start with /, and which cannot be found the directory you started from or in your home directory, it inserts \$XNEWSHOME/lib at the front of the name and tries

again.

\$XNEWSHOME/etc/News/*.ps
Startup POSTSCRIPT language code.

\$XNEWSHOME/lib/fonts/*
Font library.

\$XNEWSHOME/bin/xnews
The server.

\$XNEWSHOME/lib/openwin-init
Shell script for initializing window client applications at server start-up.

\$XNEWSHOME/lib/openwin-menu
Template for default root menu. See **buidmenu**.

~/openwin-init
User override of **\$XNEWSHOME/lib/openwin-init**

~/openwin-menu
User override of **\$XNEWSHOME/lib/openwin-menu**

~/user.ps
User-definable server customizations, loaded after other system *.ps files

~/startup.ps
User-definable server customizations, loaded before other system *.ps files.

SEE ALSO

psh(1), **pstern(1)**, **psview(1)**, **say(1)**, **journalling(1)**, **kbd_mode(1)**, **newshost(1)**, **newsserverstr(1)**, **psman(1)**, **hyperview(1)**, **buildmenu**

NeWS Programmer's Guide

POSTSCRIPT Language Reference Manual, Adobe Systems Inc., Addison-Wesley

X Window System Protocol, Version 11, MIT

Xlib - C Language X Interface, Version 11, MIT

BUGS

Some parts of the POSTSCRIPT language have yet to be implemented. See the appendix in the *NeWS Programmer's Guide* entitled *Omissions and Implementation Limits*.

The server is not yet completely robust when it runs out of memory. This out-of-memory condition occurs because swap space has been used up. Swap space is a resource that is shared by all the processes running on your machine.

When running SunView 1 programs, you may see **Window display lock broken...** messages.

If you do not have a console window, messages to the console disrupt the entire window system display.

Anyone who can gain access (legitimately or otherwise) to the system on which the server is running, or to any system allowed to access the server, has, through either the X11 or the NeWS protocol, unrestricted access to the resources of the server. They can monitor the keyboard and the mouse, read information from the screen, and interfere with the operation of other clients.

TRADEMARK

POSTSCRIPT is a registered trademark of Adobe Systems Inc.

B

X11 Manual Pages

X11 Manual Pages	133
bitmap.n	135
ico.n	143
logo.n	145
maze.n	147
muncher.n	149
plaid.n	151
pswm.n	153
puzzle.n	155
worm.n	157
xcalc.n	159
xdpr.n	163
xdpyinfo.n	165
xfd.n	167
xhost.n	169
xlsfonts.n	171
xlswins.n	173
xmac.n	175
xmag.n	177
xmodmap.n	179
xpr.n	183
xprop.n	185
xrdb.n	189



xrefresh.n	193
xset.n	195
xsetroot.n	197
xterm.n	199
xwd.n	209
xwininfo.n	211
xwud.n	213

NAME

bitmap, bmtoa, atobm – bitmap editor and converter utilities for X

SYNOPSIS

bitmap [-options ...] *filename* *WIDTHxHEIGHT*

bmtoa [-chars ...] [*filename*]

atobm [-chars *cc*] [-name *variable*] [-xhot *number*] [-yhot *number*] [*filename*].

DESCRIPTION

The *bitmap* program is a rudimentary tool for creating or editing rectangular images made up of 1's and 0's. Bitmaps are used in X for defining clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

The *bmtoa* and *atobm* filters convert *bitmap* files (FILE FORMAT) to and from ASCII strings. They are most commonly used to quickly print out bitmaps and to generate versions for including in text.

USAGE

Bitmap displays grid in which each square represents a single bit in the picture being edited. Squares can be set, cleared, or inverted directly with the buttons on the pointer and a menu of higher level operations such as draw line and fill circle is provided to the side of the grid. Actual size versions of the bitmap as it would appear normally and inverted appear below the menu.

If the bitmap is to be used for defining a cursor, one of the squares in the images may be designated as the *hotspot*. This determines where the cursor is actually pointing. For cursors with sharp tips (such as arrows or fingers), this is usually at the end of the tip; for symmetric cursors (such as crosses or bullseyes), this is usually at the center.

Bitmaps are stored as small C code fragments suitable for including in applications. They provide an array of bits as well as symbolic constants giving the width, height, and hotspot (if specified) that may be used in creating cursors, icons, and tiles.

The *WIDTHxHEIGHT* argument gives the size to use when creating a new bitmap (the default is 16x16). Existing bitmaps are always edited at their current size.

If the *bitmap* window is resized by the window manager, the size of the squares in the grid will shrink or enlarge to fit.

OPTIONS

Bitmap accepts the following options:

-help

This option will cause a brief description of the allowable options and parameters to be printed.

-display *display*

This option specifies the name of the X server to used.

-geometry *geometry*

This option specifies the placement and size of the bitmap window on the screen. See X for details.

-nodashed

This option indicates that the grid lines in the work area should not be drawn using dashed lines. Although dashed lines are prettier than solid lines, on some servers they are significantly slower.

-name *variablename*

This option specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *filename* command line argument.

-bw *number*

This option specifies the border width in pixels of the main window.

-fn *font*

This option specifies the font to be used in the buttons.

-fg *color*

This option specifies the color to be used for the foreground.

-bg *color*

This option specifies the color to be used for the background.

-hl *color*

This option specifies the color to be used for highlighting.

-bd *color*

This option specifies the color to be used for the window border.

-ms *color*

This option specifies the color to be used for the pointer (mouse).

Bmtoa accepts the following option:

-chars *cc*

This option specifies the pair of characters to use in the string version of the bitmap. The first character is used for 0 bits and the second character is used for 1 bits. The default is to use dashes (-) for 0's and sharp signs (#) for 1's.

Atobm accepts the following options:

-chars *cc*

This option specifies the pair of characters to use when converting string bitmaps into arrays of numbers. The first character represents a 0 bit and the second character represents a 1 bit. The default is to use dashes (-) for 0's and sharp signs (#) for 1's.

-name *variable*

This option specifies the variable name to be used when writing out the bitmap file. The default is to use the basename of the *filename* command line argument or leave it blank if the standard input is read.

-xhot *number*

This option specifies the X coordinate of the hotspot. Only positive values are allowed. By default, no hotspot information is included.

-yhot *number*

This option specifies the Y coordinate of the hotspot. Only positive values are allowed. By default, no hotspot information is included.

CHANGING GRID SQUARES

Grid squares may be set, cleared, or inverted by pointing to them and clicking one of the buttons indicated below. Multiple squares can be changed at once by holding the button down and dragging the cursor across them. Set squares are filled and represent 1's in the bitmap; clear squares are empty and represent 0's.

Button 1

This button (usually leftmost on the pointer) is used to set one or more squares. The corresponding bit or bits in the bitmap are turned on (set to 1) and the square or squares are filled.

Button 2

This button (usually in the middle) is used to invert one or more squares. The corresponding bit or bits in the bitmap are flipped (1's become 0's and 0's become 1's).

Button 3

This button (usually on the right) is used to clear one or more squares. The corresponding bit or bits in the bitmap are turned off (set to 0) and the square or squares are emptied.

MENU COMMANDS

To make defining shapes easier, *bitmap* provides 13 commands for drawing whole sections of the grid at once, 2 commands for manipulating the hotspot, and 2 commands for updating the bitmap file and exiting. A command buttons for each of these operations is located to the right of the grid.

Several of the commands operate on rectangular portions of the grid. These areas are selected after the command button is pressed by moving the cursor to the upper left square of the desired area, pressing a pointer button, dragging the cursor to the lower right hand corner (with the button still pressed) , and then releasing the button. The command may be aborted by pressing any other button while dragging or by releasing outside the grid.

To invoke a command, move the pointer over that command and click any button.

Clear All

This command is used to clear all of the bits in the bitmap as if Button 3 had been dragged through every square in the grid. It cannot be undone.

Set All

This command is used to set all of the bits in the bitmap as if Button 1 had been dragged through every square in the grid. It cannot be undone.

Invert All

This command is used to invert all of the bits in the bitmap as if Button 2 had been dragged through every square in the grid.

Clear Area

This command is used to clear a region of the grid as if Button 3 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be cleared should be selected as outlined above.

Set Area

This command is used to set a region of the grid as if Button 1 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be set should be selected as outlined above.

Invert Area

This command is used to inverted a region of the grid as if Button 2 had been dragged through each of the squares in the region. When this command is invoked, the cursor will change shape to indicate that the area to be inverted should be selected as outlined above.

Copy Area

This command is used to copy a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be copied should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be copied.

Move Area

This command is used to move a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be moved should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be moved. Any squares in the region's old position that aren't also in the new position are cleared.

Overlay Area

This command is used to copy all of the set squares in a region of the grid from one location to another. When this command is invoked, the cursor will change shape to indicate that the area to be copied should be selected as outlined above. The cursor should then be clicked on the square to which the upper left hand corner of the region should be overlaid. Only the squares that are set in the region will be touched in the new location.

Line

This command will set the squares in a line between two points. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the two end points of the line.

Circle

This command will set the squares on a circle specified by a center and a point on the curve. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the center of the circle and then over a point on the curve. Small circles may not look very round because of the size of the grid and the limits of having to work with discrete pixels.

Filled Circle

This command will set all of the squares in a circle specified by a center and a point on the curve. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on the center of the circle and then over a point on the curve. All squares side and including the circle are set.

Flood Fill

This command will set all clear squares in an enclosed shape. When this command is invoked, the cursor will change shape to indicate that the pointer should be clicked on any empty square inside the shape to be filled. All empty squares that border horizontally or vertically with the indicated square are set out to the enclosing shape. If the shape is not closed, the entire grid will be filled.

Set Hot Spot

This command designates one square in the grid as the hot spot if this bitmap to be used for defining a cursor. When the command is invoked, the cursor will change indicating that the pointer should be clicked on the square to contain the hot spot.

Clear Hot Spot

This command removes any designated hot spot from the bitmap.

Write Output

This command writes a small fragment of C code representing the bitmap to the filename specified on the command line. If the file already exists, the original file will be renamed to *filename~* before the new file is created. If an error occurs in either the renaming or the writing of the bitmap file, a dialog box will appear asking whether or not *bitmap* should use *tmpfilename* instead.

Quit

This command causes *bitmap* to display a dialog box asking whether or not it should save the bitmap (if it has changed) and then exit. Answering *yes* is the same as invoking *Write Output*; *no* causes *bitmap* to simply exit; and *cancel* will abort the *Quit* command so that more changes may be made.

FILE FORMAT

The *Write Output* command stores bitmaps as simple C program fragments that can be compiled into programs, referred to by X Toolkit pixmap resources, manipulated by other programs (see *xsetroot*), or read in using utility routines in the various programming libraries. The width and height of the bitmap as well as the hotspot, if specified, are written as preprocessor symbols at the start of the file. The bitmap image is then written out as an array of characters:

```
#define name_width 11
#define name_height 5
#define name_x_hot 5
#define name_y_hot 2

static char name_bits[] = {
```

```

    0x91, 0x04, 0xca, 0x06, 0x84,
    0x04, 0x8a, 0x04, 0x91, 0x04
};

```

The **name** prefix to the preprocessor symbols and to the bits array is constructed from the *filename* argument given on the command line. Any directories are stripped off the front of the name and any suffix beginning with a period is stripped off the end. Any remaining non-alphabetic characters are replaced with underscores. The *name_x_hot* and *name_y_hot* symbols will only be present if a hotspot has been designated using the *Set Hot Spot* command.

Each character in the the array contains 8 bits from one row of the image (rows are padded out at the end to a multiple of 8 to make this is possible). Rows are written out from left to right and top to bottom. The first character of the array holds the leftmost 8 bits of top line, and the last characters holds the right most 8 bits (including padding) of the bottom line. Within each character, the leftmost bit in the bitmap is the least significant bit in the character.

This process can be demonstrated visually by splitting a row into words containing 8 bits each, reversing the bits each word (since Arabic numbers have the significant digit on the right and images have the least significant bit on the left), and translating each word from binary to hexadecimal.

In the following example, the array of 1's and 0's on the left represents a bitmap containing 5 rows and 11 columns that spells *X11*. To its right is is the same array split into 8 bit words with each row padded with 0's so that it is a multiple of 8 in length (16):

```

    10001001001    10001001 00100000
    01010011011    01010011 01100000
    00100001001    00100001 00100000
    01010001001    01010001 00100000
    10001001001    10001001 00100000

```

Reversing the bits in each word of the padded, split version of the bitmap yields the left hand figure below. Interpreting each word as hexadecimal number yields the array of numbers on the right:

```

    10010001 00000100    0x91 0x04
    11001010 00000110    0xca 0x06
    10000100 00000100    0x84 0x04
    10001010 00000100    0x8a 0x04
    10010001 00000100    0x91 0x04

```

The character array can then be generated by reading each row from left to right, top to bottom:

```

static char name_bits[] = {
    0x91, 0x04, 0xca, 0x06, 0x84,
    0x04, 0x8a, 0x04, 0x91, 0x04
};

```

The *bmtoa* program may be used to convert *bitmap* files into arrays of characters for printing or including in text files. The *atobm* program can be used to convert strings back to *bitmap* format.

USING BITMAPS IN PROGRAMS

The format of *bitmap* files is designed to make bitmaps and cursors easy to use within X programs. The following code could be used to create a cursor from bitmaps defined in *this.cursor* and *this_mask.cursor*:

```

#include "this.cursor"
#include "this_mask.cursor"

```

```

XColor foreground, background;
/* fill in foreground and background color structures */
Pixmap source = XCreateBitmapFromData (display, drawable,
    this_bits, this_width, this_height);
Pixmap mask = XCreateBitmapFromData (display, drawable,
    this_mask_bits, this_mask_width, this_mask_height);
Cursor cursor = XCreatePixmapCursor (display, source, mask,
    foreground, background, this_x_hot, this_y_hot);

```

Additional routines are available for reading in *bitmap* files and returning the data in the file, in Bitmap (single-plane Pixmap for use with routines that require stipples), or full depth Pixmap (often used for window backgrounds and borders). Applications writers should be careful to understand the difference between Bitmaps and Pixmap so that their programs function correctly on color and monochrome displays.

For backward compatibility, *bitmap* will also accept X10 format *bitmap* files. However, when the file is written out again it will be in X11 format

X DEFAULTS

Bitmap uses the following resources:

Background

The window's background color. Bits which are 0 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is *white*.

BorderColor

The border color. This option is useful only on color displays. The default value is *black*.

BorderWidth

The border width. The default value is 2.

BodyFont

The text font. The default value is *variable*.

Dashed

If "off", then *bitmap* will draw the grid lines with solid lines. The default is "on".

Foreground

The foreground color. Bits which are 1 in the bitmap are displayed in this color. This option is useful only on color displays. The default value is *black*.

Highlight

The highlight color. *bitmap* uses this color to show the hot spot and to indicate rectangular areas that will be affected by the *Move Area*, *Copy Area*, *Set Area*, and *Invert Area* commands. If a highlight color is not given, then *bitmap* will highlight by inverting. This option is useful only on color displays.

Mouse

The pointer (mouse) cursor's color. This option is useful only on color displays. The default value is *black*.

Geometry

The size and location of the bitmap window.

Dimensions

The *WIDTHxHEIGHT* to use when creating a new bitmap.

SEE ALSO

X(1), *Xlib - C Language X Interface* (particularly the section on *Manipulating Bitmaps*), *XmuReadBitmap-DataFromFile*

BUGS

The old command line arguments aren't consistent with other X programs.

If you move the pointer too fast while holding a pointer button down, some squares may be missed. This is caused by limitations in how frequently the X server can sample the pointer location.

There is no way to write to a file other than the one specified on the command line.

There is no way to change the size of the bitmap once the program has started.

There is no *undo* command.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

bitmap by Ron Newman, MIT Project Athena; documentation, *bmtoa*, and *atobm* by Jim Fulton, MIT X Consortium.

NAME

ico – animate an icosahedron

SYNOPSIS

ico [display list] [=geometry] [-d pattern] [-i]

DESCRIPTION

Ico displays a wire-frame rotating polyhedron, with hidden lines removed, or a solid-fill polyhedron with hidden faces removed. There are a number of different polyhedra available; adding a new polyhedron to the program is quite simple.

OPTIONS**-d pattern**

Specify a bit pattern for drawing dashed lines for wire frames.

-i Use inverted colors for wire frames.

For each display specified,

Ico creates another window. The total geometry is split vertically amongst these windows, and the *ico* bounces between them. The end result is that **ico foo:0 bar:0** would result in *ico* bouncing half on display **foo:0** and half on **bar:0**.

SEE ALSO

X(1)

BUGS

Doesn't deal too well with being resized.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.



NAME

XNEWSLOGO – displays the X11 and NeWS logos.

SYNOPSIS

XCOLOR [*-display display*] [*-geometry geometry*]

DESCRIPTION

XNEWSLOGO displays the MIT X11 logo and the Sun Microsystems logo in a single X window. The X11 logo is rendered using the X11 code from the Xtk widget set's "logo widget". The Sun Logo is rendered using postscript code being sent to the NeWS interpreter in the X11/NeWS merge.

OPTIONS

-display connection

Connect to X server display, *connection*.

-geometry geomspec

Set window size and placement to the standard X11 geometry specification, *geomspec*.

SEE ALSO

X(1)

COPYRIGHT

Copyright (c) 1988 by Sun Microsystems, Inc.
David Lemke (lemke@wirehead.sun.com)

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NAME

maze - an automated maze program... [demo][X11]

SYNTAX

maze [-S] [-r] [-g *geometry*] [-d *display*]

DESCRIPTION

The *maze* program creates a "random" maze and then solves it with graphical feedback.

Command Options

-S Full screen window option...

-r Reverse video option...

-g *geometry*
Specifies the window geometry to be used...

-d *display*
Specifies the display to be used...

The following lists the current functionality of various mouse button clicks;

LeftButton

Clears the window and restarts maze...

MiddleButton

Toggles the maze program, first click -> *stop*, second click -> *continue*...

RightButton

Kills maze...

LIMATIONS

No color support...

Expose events force a restart of maze...

Currently, mouse actions are based on "raw" values [Button1, Button2 and Button3] from the ButtonPress event...

[doesn't use pointer mapping]

COPYRIGHT

Copyright 1988 by Sun Microsystems, Inc. Mountain View, CA.

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation, and that the names of Sun or MIT not be used in advertising or publicity pertaining to distribution of the software without specific prior written permission. Sun and M.I.T. make no representations about the suitability of this software for any purpose. It is provided "as is" without any express or implied warranty.

SUN DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL SUN BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

AUTHOR(s)

Richard Hess	[X11 extensions]	{...}!uunet!cimshop!rhess
Consilium, Mountain View, CA		
Dave Lemke	[X11 version]	lemke@sun.COM

Sun Microsystems, Mountain View, CA
Martin Weiss [SunView version]
Sun Microsystems, Mountain View, CA

NAME

muncher – draw interesting patterns in an X window

SYNOPSIS

muncher [-option ...]

OPTIONS

- r** display in the root window
- s *seed*** seed the random number seed
- v** run in verbose mode
- q** run in quite mode
- geometry *geometry***
define the initial window geometry; see *X(1)*.
- display *display***
specify the display to use; see *X(1)*.

DESCRIPTION

Muncher draws some interesting patterns in a window.

SEE ALSO

X(1)

BUGS

There are no known bugs. There are lots of lacking features.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.



NAME

plaid – paint some plaid-like patterns in an X window

SYNOPSIS

plaid [-option ...]

OPTIONS

- b** enable backing store for the window
- geometry *geometry***
define the initial window geometry; see *X(1)*.
- display *display***
specify the display to use; see *X(1)*.

DESCRIPTION

Plaid displays a continually changing plaid-like pattern in a window.

SEE ALSO

X(1)

BUGS

There are no known bugs. There are lots of lacking features.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

NAME

pswm – POSTSCRIPT-based X11 window manager

SYNOPSIS

pswm [**-display** *string*] [**-helpkey** *keycode*] [**-init**]

DESCRIPTION

pswm is an X11 window manager for use with the X11/NeWS server. **pswm** manages X11 windows using the NeWS toolkit. This enables X11 and NeWS windows to coexist on the same screen.

STARTUP AND SHUTDOWN

The X11/NeWS server starts **pswm** automatically by default. The server provides the **-init** option so that **pswm** will execute a startup script.

You will need to shut down **pswm** if you want to run a different X11 window manager. To shut down **pswm**, simply send it a TERM signal using **kill(1)**. You must have **pswm**'s process id before you can do this. You can find it by using the **ps(1)** command. For example:

```
example% ps ax | grep pswm
 7840 p5 S      0:00 grep pswm
 7710 p6 IW     0:00 pswm -init
example% kill -TERM 7710
```

To restart **pswm**, simply run it from a shell, in the background.

```
example% pswm &
```

OPTIONS**-display** *string*

Causes **pswm** to connect to the X11 display identified by *string*. *string* has the same format as the **DISPLAY** environment variable. If the **-display** option is present, *string* will override any value in the **DISPLAY** environment variable, if one is present.

-helpkey *keycode*

Specifies the keycode of the key which is to be used as the *Help* key.

-init

Causes **pswm** to run a startup script. **pswm** first tries to execute “**\$HOME**/.openwin-init”. If this fails, **pswm** will then try to execute a default startup script from “**\$XNEWSHOME**/lib/openwin-init”.

FILES

\$HOME/.openwin-init User-customizable startup script.

\$XNEWSHOME/lib/openwin-init Default startup script.

DIAGNOSTICS

can't open X11 (or NeWS) connection

pswm needs to create an X11 and a NeWS connection to the same server. It failed to make one of the required server connections.

can't rendezvous X11 and NeWS connections

pswm's needs to create an X11 and a NeWS connection to the same server. These connections ended up at different servers. Check the **DISPLAY** and **NEWSERVER** environment variables to make sure they indicate the same X11/NeWS server.

another window manager is already running on this screen

You can run only one window manager at a time on a given screen. There's another window manager running already, so **pswm** can't run.

COPYRIGHT

Copyright (c) 1989 by Sun Microsystems, Inc.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

TRADEMARKS

PostScript is a trademark of Adobe Systems, Inc.

NAME

puzzle – 15-puzzle game for X

SYNOPSIS

puzzle [-option ...]

OPTIONS

–display *display*

This option specifies the display to use; see *X(1)*.

–geometry *geometry*

This option specifies the size and position of the puzzle window; see *X(1)*.

–size *WIDTHxHEIGHT*

This option specifies the size of the puzzle in squares.

–speed *num*

This option specifies the speed in tiles per second for moving tiles around.

–picture *filename*

This option specifies an image file containing the picture to use on the tiles. Try “mandrill.cm.”

This only works on 8-bit pseudo-color screens.

–colormap

This option indicates that the program should create its own colormap for the picture option.

DESCRIPTION

Puzzle with no arguments plays a 4x4 15-puzzle. The control bar has two boxes in it. Clicking in the left box scrambles the puzzle. Clicking in the right box solves the puzzle. Clicking the middle button anywhere else in the control bar causes puzzle to exit. Clicking in the tiled region moves the empty spot to that location if the region you click in is in the same row or column as the empty slot.

SEE ALSO

X(1)

BUGS

The picture option should work on a wider variety of screens.

COPYRIGHT

Copyright 1988, Don Bennett.

AUTHOR

Don Bennett, HP Labs

NAME

worm – draw wiggly worms

SYNOPSIS

worm [*-l length*] [*-s size*] [*-n number*] [*-d connection*] [*-g geometry*] [*-R*] [*-C*] [*-S*]

DESCRIPTION

worm draws wiggly worms. It is adapted from a concept in the December 1987 issue of Scientific American. Playing with the various parameters can create strange effects. Pressing any key in the worm window will cause them to freeze; pressing again will thaw.

OPTIONS

- S* Screensaver. Takes over entire screen.
- C* Chromocolor. Worms change colors as they crawl.
- R* Rotate colormap. The colormap constantly changes.
- n number*
Make *number* worms. Default is 50.
- l length*
Worms are *length* pixels long. A negative value means infinite length.
- size size*
Worms are *size* pixels wide.
- display connection*
Connect to X server display, *connection*.
- geometry geomspec*
Create window using *geomspec*.

SEE ALSO

X(1)

COPYRIGHT

Copyright (c) 1988 by Sun Microsystems, Inc.
David Lemke (lcmke@wirehead.sun.com)

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.



NAME

xcalc – scientific calculator for X

SYNOPSIS

xcalc [-display *display*] [-bw *pixels*] [-stip] [-rv] [-rpn] [-analog] [-geometry *geometry*]

DESCRIPTION

Xcalc is a scientific calculator desktop accessory that can emulate a TI-30, an HP-10C, and a slide rule.

OPTIONS

–display *displayname*

This option specifies the X server to contact.

–geometry *geometry*

This option specifies the size and placement of the top level window. By default, the minimum size will be used. Note that your window manager may require you to place it explicitly anyway.

–fg *color*

This option specifies the foreground color to use.

–bg *color*

This option specifies the background color to use.

–bw *pixels*

This option specifies the border width in pixels.

–stip

This option indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays this makes for a nicer display.

–rv

This option indicates that reverse video should be used.

–rpn

This option indicates that Reverse Polish Notation should be used. In this mode the calculator will look and behave like an HP-10C. Without this flag, it will emulate a TI-30.

–analog This option indicates that a slide rule should be used.

OPERATION

Pointer Usage: Most operations are done with the Button1 (usually leftmost button on the pointer). The only exception is that pressing the AC key on the TI calculator with Button3 (usually on the right) will exit the calculator.

Key Usage (Normal mode): The number keys, the +/- key, and the +, -, *, /, and = keys all do exactly what you would expect them to. It should be noted that the operators obey the standard rules of precedence. Thus, entering "3+4*5=" results in "23", not "35". The parentheses can be used to override this. For example, "(1+2+3)*(4+5+6)=" results in "6*15=90". The non-obvious keys are detailed below.

1/x replaces the number in the display with its reciprocal.

x^2 squares the number in the display.

SQRT takes the square root of the number in the display.

CE/C when pressed once, clears the number in the display without clearing the state of the machine. Allows you to re-enter a number if you screw it up. Pressing it twice clears the state, also.

AC clears everything, the display, the state, the memory, everything. Pressing it with the right button 'turns off' the calculator, in that it exits the program. Somewhat more equivalent to throwing the calculator in the trash, if we were to pursue the analogy.

INV inverts the meaning of the function keys. See the individual function keys for details.

sin computes the sine of the number in the display, as interpreted by the current DRG mode (see DRG, below). If inverted, it computes the arcsine.

cos computes the cosine, or arccosine when inverted.

tan computes the tangent, or arctangent when inverted.

DRG changes the DRG mode, as indicated by 'DEG', 'RAD', or 'GRAD' at the bottom of number window of the calculator. When in 'DEG' mode, numbers in the display are taken as being degrees. In 'RAD' mode, numbers are in radians, and in 'GRAD' mode, numbers are in gradians. When inverted, the DRG key has the nifty feature of converting degrees to radians to gradians and vice-versa. Example: put the calculator into 'DEG' mode, and type "45 INV DRG". The display should now show something along the lines of ".785398", which is 45 degrees converted to radians.

e the constant 'e'. (2.7182818...)

EE used for entering exponential numbers. For example, to enter "-2.3E-4" you'd type "2 . 3 +/- EE 4 +/-"

log calculates the log (base 10) of the number in the display. When inverted, it raises "10.0" to the number in the display. For example, typing "3 INV log" should result in "1000".

ln calculates the log (base e) of the number in the display. When inverted, it raises "e" to the number in the display. For example, typing "e ln" should result in "1"

y^x raises the number on the left to the power of the number on the right. For example "2 y^x 3 =" results in "8", which is 2³. For a further example, "(1+2+3) y^x (1+2) =" equals "6 y^x 3" which equals "216".

PI the constant 'pi'. (3.1415927....)

x! computes the factorial of the number in the display. The number in the display must be an integer in the range 0-500, though, depending on your math library, it might overflow long before that.

STO copies the number in the display to the memory location.

RCL copies the number from the memory location to the display.

SUM adds the number in the display to the number in the memory location.

EXC swaps the number in the display with the number in the memory location.

Key Usage (RPN mode): The number keys, CHS (change sign), +, -, *, /, and ENTR keys all do exactly what you would expect them to do. Many of the remaining keys are the same as in normal mode. The differences are detailed below.

<- is a backspace key that can be used while typing a number. It will erase digits from the display.

ON clears everything, the display, the state, the memory, everything. Pressing it with the right button 'turns off' the calculator, in that it exits the program. Somewhat more equivalent to throwing the calculator in the trash, if we were to pursue the analogy.

INV inverts the meaning of the function keys. This would be the "f" key on an HP calculator, but xcalc does not have the resolution to display multiple legends on each key. See the individual function keys for details.

10^x raises "10.0" to the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display.

e^x raises "e" to the number in the top of the stack. When inverted, it calculates the log (base e) of the number in the display.

STO copies the number in the top of the stack to a memory location. There are 10 memory locations. The desired memory is specified by following this key with pressing a digit key.

RCL pushes the number from the specified memory location onto the stack.

SUM adds the number on top of the stack to the number in the specified memory location.

x:y exchanges the numbers in the top two stack positions.

R v rolls the stack downward. When inverted, it rolls the stack upward.

blank these keys were used for programming functions on the HP11-C. Their functionality has not been duplicated here.

KEYBOARD EQUIVALENTS

If you have the pointer in the *xcalc* window, you can use the keyboard to speed entry, as almost all of the calculator keys have a keyboard equivalent. The number keys, the operator keys, and the parentheses all have the obvious equivalent. The less-obvious equivalents are as follows:

n: +/-	!: x!
p: PI	e: EE
l: ln	^: y^x
i: INV	s: sin
c: cos	t: tan
d: DRG	BS, DEL: CE/C (" \leftarrow " in RPN mode)
CR: ENTR	q: quit

COLOR USAGE

Xcalc uses a lot of colors, given the opportunity. In the default case, it will just use two colors (Foreground and Background) for everything. This works out nicely. However, if you're a color fanatic you can specify the colors used for the number keys, the operator (+-*/=) keys, the function keys, the display, and the icon.

X DEFAULTS

The program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

BorderWidth

Specifies the width of the border. The default is 2.

ReverseVideo

Indicates that reverse video should be used.

Stipple Indicates that the background should be stippled. The default is "on" for monochrome displays, and "off" for color displays.

Mode Specifies the default mode. Allowable values are *rpn*, *analog*.

Foreground

Specifies the default color used for borders and text.

Background

Specifies the default color used for the background.

NKeyFore, NKeyBack

Specifies the colors used for the number keys.

OKeyFore, OKeyBack

Specifies the colors used for the operator keys.

FKeyFore, FKeyBack

Specifies the colors used for the function keys.

DispFore, DispBack

Specifies the colors used for the display.

IconFore, IconBack

Specifies the colors used for the icon.

EXAMPLES

If you're running on a monochrome display, you shouldn't need any *.Xdefaults* entries for *xcalc*. On a color display, you might want to try the following in normal mode:

xcalc.Foreground:	Black
xcalc.Background:	LightSteelBlue
xcalc.NKeyFore:	Black
xcalc.NKeyBack:	White
xcalc.OKeyFore:	Aquamarine
xcalc.OKeyBack:	DarkSlateGray
xcalc.FKeyFore:	White
xcalc.FKeyBack:	#900
xcalc.DispFore:	Yellow
xcalc.DispBack:	#777
xcalc.IconFore:	Red
xcalc.IconBack:	White

SEE ALSO

X(1), xrdp(1)

BUGS

The calculator doesn't resize.

The slide rule and HP mode may or may not work correctly.

This application should really be implemented with the X Toolkit. It would make a very good example of a compound widget.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHORS

John Bradley, University of Pennsylvania
Mark Rosenstein, MIT Project Athena

NAME

`xdpr` - dump an X window directly to a printer

SYNOPSIS

`xdpr [filename] [-display host:display] [-Pprinter] [-device printer_device] [option ...]`

DESCRIPTION

Xdpr uses the commands *xwd*(1), *xpr*(1), and *lpr*(1) to dump an X window, process it for a particular printer type, and print it out on the printer of your choice. This is the easiest way to get a printout of a window. *Xdpr* by default will print the largest possible representation of the window on the output page.

The options for *xdpr* are the same as those for *xpr*, *xwd*, and *lpr*. The most commonly-used options are described below; see the manual pages for these commands for more detailed descriptions of the many options available.

filename

Specifies a file containing a window dump (created by *xwd*) to be printed instead of selecting an X window.

-Pprinter

Specifies a printer to send the output to. If a printer name is not specified here, *xdpr* (really, *lpr*) will send your output to the printer specified by the *PRINTER* environment variable. Be sure that type of the printer matches the type specified with the *-device* option.

-display host:display[.screen]

Normally, *xdpr* gets the host and display number to use from the environment variable "DISPLAY". One can, however, specify them explicitly; see *X*(1).

-device printer-device

Specifies the device type of the printer. Available printer devices are "ln03" for the DEC LN03, "pp" for the IBM 3812 PagePrinter, and "ps" for any postscript printer (e.g. DEC LN03R or LPS40). The default is "ln03".

-help This option displays the list of options known to *xdpr*.

Any other arguments will be passed to the *xwd*(1), *xpr*(1), and *lpr*(1) commands as appropriate for each.

SEE ALSO

xwd(1), *xpr*(1), *lpr*(1), *xwud*(1), *X*(1)

ENVIRONMENT

DISPLAY - for which display to use by default.

PRINTER - for which printer to use by default.

COPYRIGHT

Copyright 1985, 1988, Massachusetts Institute of Technology.

See *X*(1) for a full statement of rights and permissions.

AUTHOR

Paul Boutin, MIT Project Athena

Michael R. Gretzinger, MIT Project Athena

Jim Gettys, MIT Project Athena

NAME

`xcpyinfo` - display information utility for X

SYNOPSIS

`xcpyinfo` [-display *displayname*]

DESCRIPTION

Xcpyinfo is a utility for displaying information about an X server. It is used to examine the capabilities of a server, the predefined values for various parameters used in communicating between clients and the server, and the different types of screens and visuals that are available.

EXAMPLE

The following shows a sample produced by *xcpyinfo* when connected to display that supports an 8 plane Pseudocolor screen as well as a 1 plane (monochrome) screen.

```

name of display:  empire:0.0
version number:  11.0
vendor string:   MIT X Consortium
vendor release number:  3
maximum request size: 16384 longwords (65536 bytes)
motion buffer size:  0
bitmap unit, bit order, padding:  32, MSBFirst, 32
image byte order:  MSBFirst
number of supported pixmap formats: 2
supported pixmap formats:
    depth 1, bits_per_pixel 1, scanline_pad 32
    depth 8, bits_per_pixel 8, scanline_pad 32
keycode range:  minimum 8, maximum 129
default screen number: 0
number of screens: 2

screen #0:
dimensions: 1152x900 pixels (325x254 millimeters)
resolution: 90x90 dots per inch
root window id: 0x8006d
depth of root window: 1 plane
number of colormaps: minimum 1, maximum 1
default colormap: 0x80065
default number of colormap cells: 2
preallocated pixels: black 1, white 0
options: backing-store YES, save-unders YES
current input event mask: 0x1b8003c
    ButtonPressMask      ButtonReleaseMask      EnterWindowMask
    LeaveWindowMask      SubstructureNotifyMask  SubstructureRedirectMask
    FocusChangeMask      ColormapChangeMask      OwnerGrabButtonMask
number of visuals: 1
default visual id: 0x80064
visual:
    visual id: 0x80064
    class: StaticGray
    depth: 1 plane
    size of colormap: 2 entries
    red, green, blue masks: 0x0, 0x0, 0x0
    significant bits in color specification: 1 bits

screen #1:

```

dimensions: 1152x900 pixels (325x254 millimeters)
resolution: 90x90 dots per inch
root window id: 0x80070
depth of root window: 8 planes
number of colormaps: minimum 1, maximum 1
default colormap: 0x80067
default number of colormap cells: 256
preallocated pixels: black 1, white 0
options: backing-store YES, save-unders YES
current input event mask: 0x0
number of visuals: 1
default visual id: 0x80066
visual:
 visual id: 0x80066
 class: PseudoColor
 depth: 8 planes
 size of colormap: 256 entries
 red, green, blue masks: 0x0, 0x0, 0x0
 significant bits in color specification: 8 bits

ENVIRONMENT**DISPLAY**

To get the default host, display number, and screen.

SEE ALSO

X(1), xwininfo(1), xprop(1), xrdb(1)

BUGS

Due to a bug in the Xlib interface, there is currently no portable way to determine the depths of pixmap images that are supported by the server.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X(1)* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

`xfd` - font displayer for X

SYNOPSIS

`xfd` [-options ...] -fn *fontname*

OPTIONS

- display** *display*
Specifies the display to use.
- geometry** *geometry*
Specifies an initial window geometry.
- bw** *number*
Allows you to specify the width of the window border in pixels.
- rv**
The foreground and background colors will be switched. The default colors are black on white.
- fw**
Overrides a previous choice of reverse video. The foreground and background colors will not be switched.
- fg** *color*
On color displays, determines the foreground color (the color of the text).
- bg** *color*
On color displays, determines the background color.
- bd** *color*
On color displays, determines the color of the border.
- bf** *fontname*
Specifies the font to be used for the messages at the bottom of the window.
- tl** *title*
Specifies that the title of the displayed window should be *title*.
- in** *iconname*
Specifies that the name of the icon should be *iconname*.
- icon** *filename*
Specifies that the bitmap in file *filename* should be used for the icon.
- verbose**
Specifies that extra information about the font should be displayed.
- gray**
Specifies that a gray background should be used.
- start** *charnum*
Specifies that character number *charnum* should be the first character displayed.

DESCRIPTION

Xfd creates a window in which the characters in the named font are displayed. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left will be character number 0 unless the -start option has been supplied in which case the character with the number given in the -start option will be used.

The characters are displayed in a grid of boxes, each large enough to hold any single character in the font. If the -gray option has been supplied, the characters will be displayed using XDrawImageString using the foreground and background colors on a gray background. This permits determining exactly how XDrawImageString will draw any given character. If -gray has not been supplied, the characters will simply be drawn using the foreground color on the background color.

All the characters in the font may not fit in the window at once. To see additional characters, click the right mouse button on the window. This will cause the next window full of characters to be displayed. Clicking the left mouse button on the window will cause the previous window full of characters to be displayed. *Xfd* will beep if an attempt is made to go back past the 0th character.

Note that if the font is a 8 bit font, the characters 256-511 (100-1ff in hexadecimal), 512-767 (200-2ff in hexadecimal), ... will display exactly the same as the characters 0-255 (00-ff in hexadecimal). *Xfd* by default creates a window big enough to display 16 rows of 16 columns (totally 256 characters).

Clicking the middle button on a character will cause that character's number to be displayed in both decimal and hexadecimal at the bottom of the window. If verbose mode is selected, additional information about that particular character will be displayed as well. The displayed information includes the width of the character, its left bearing, right bearing, ascent, and its descent. If verbose mode is selected, typing '<' or '>' into the window will display the minimum or maximum values respectively taken on by each of these fields over the entire font.

The font name is interpreted by the X server. To obtain a list of all the fonts available, use *xlsfonts(1)*.

The window stays around until the *xfd* process is killed or one of 'q', 'Q', ' ', or ctrl-c is typed into the *Xfd* window.

X DEFAULTS

Xfd uses the following X resources:

BorderWidth	Set the border width of the window.
BorderColor	Set the border color of the window.
ReverseVideo	If "on", reverse the definition of foreground and background color.
Foreground	Set the foreground color.
Background	Set the background color.
BodyFont	Set the font to be used in the body of the window. (I.e., for messages, etc.) This is not the font that <i>Xfd</i> displays, just the font it uses to display information about the font being displayed.
IconName	Set the name of the icon.
IconBitmap	Set the file we should look in to get the bitmap for the icon.
Title	Set the title to be used.

SEE ALSO

X(1), *xlsfonts(1)*, *xrdb(1)*

BUGS

It should display the name of the font somewhere.

Character information displayed in verbose mode is sometimes clipped to the window boundary, hiding it from view.

It should be rewritten to use the X toolkit.

It should skip over pages full of non-existent characters.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xhost - server access control program for X

SYNOPSIS

xhost [[+]-]hostname ...]

DESCRIPTION

The *xhost* program is used to add and delete hosts to the list of machines that are allowed to make connections to the X server. This provides a rudimentary form of privacy control and security. It is only sufficient for a workstation (single user) environment, although it does limit the worst abuses. Environments which require more sophisticated measures should use the hooks in the protocol for passing authentication data to the server.

The server initially allows network connections only from programs running on the same machine or from machines listed in the file */etc/X*.hosts* (where * is the display number of the server). The *xhost* program is usually run either from a startup file or interactively to give access to other users.

Hostnames that are followed by two colons (::) are used in checking DECnet connections; all other hostnames are used for TCP/IP connections.

OPTIONS

Xhost accepts the following command line options described below. For security, the options that effect access control may only be run from the same machine as the server.

[+]hostname

The given *hostname* (the plus sign is optional) is added to the list of machines that are allowed to connect to the X server.

-hostname

The given *hostname* is removed from the list of machines that are allowed to connect to the server. Existing connections are not broken, but new connection attempts will be denied. Note that the current machine is allowed to be removed; however, further connections (including attempts to add it back) will not be permitted. Resetting the server (thereby breaking all connections) is the only way to allow local connections again.

+ Access is granted to everyone, even if they aren't on the list of allowed hosts (i.e. access control is turned off).

- Access is restricted to only those machines on the list of allowed hosts (i.e. access control is turned on).

nothing If no command line arguments are given, the list of hosts that are allowed to connect is printed on the standard output along with a message indicating whether or not access control is currently enabled. This is the only option that may be used from machines other than the one on which the server is running.

FILES

/etc/X.hosts*

SEE ALSO

X(1), Xserver(1)

ENVIRONMENT**DISPLAY**

to get the default host and display to use.

BUGS

You can't specify a display on the command line because **-display** is a valid command line argument (indicating that you want to remove the machine named "*display*" from the access list).

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See X(1) for a full statement of rights and permissions.

AUTHORS

Bob Scheifler, MIT Laboratory for Computer Science,
Jim Gettys, MIT Project Athena (DEC).

NAME

xlsfonts - server font list displayer for X

SYNOPSIS

xlsfonts [-options ...] [-fn pattern]

DESCRIPTION

Xlsfonts lists the fonts that match the given *pattern*. The wildcard character "*" may be used to match any sequence of characters (including none), and "?" to match any single character. If no pattern is given, "*" is assumed.

The "*" and "?" characters must be quoted to prevent them from being expanded by the shell.

OPTIONS

-display *host:dpy*

This option specifies the X server to contact.

-l This option indicates that a long listing should be generated for each font.

-L This option indicates that a very long listing showing the individual character metrics should be printed.

-m This option indicates that long listings should also print the minimum and maximum bounds of each font.

-C This option indicates that listings should use multiple columns. This is the same as -n 0.

-1 This option indicates that listings should use a single column. This is the same as -n 1.

-w *width*

This option specifies the width in characters that should be used in figuring out how many columns to print. The default is 79.

-n *columns*

This option specifies the number of columns to use in displaying the output. By default, it will attempt to fit as many columns of font names into the number of character specified by -w *width*.

SEE ALSO

X(1), Xserver(1), xset(1), xfd(1)

ENVIRONMENT**DISPLAY**

to get the default host and display to use.

BUGS

Doing "xlsfonts -l" can tie up your server for a very long time. This is really a bug with single-threaded non-preemptible servers, not with this program.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See X(1) for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xlswins - server window list displayer for X

SYNOPSIS

xlswins [-options ...] [windowid ...]

DESCRIPTION

Xlswins lists the window tree. By default, the root window is used as the starting point, although a specific window may be specified using the *-id* option. If no specific windows are given on the command line, the root window will be used.

OPTIONS

-display *displayname*

This option specifies the X server to contact.

-l This option indicates that a long listing should be generated for each window. This includes a number indicating the depth, the geometry relative to the parent as well as the location relative to the root window.

-format *radix*

This option specifies the radix to use when printing out window ids. Allowable values are: *hex*, *octal*, and *decimal*. The default is hex.

-indent *number*

This option specifies the number of spaces that should be indented for each level in the window tree. The default is 2.

SEE ALSO

X(1), Xserver(1), xwininfo(1), xprop(1)

ENVIRONMENT

DISPLAY

to get the default host and display to use.

BUGS

This should be integrated with xwininfo somehow.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See X(1) for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

xmac – display Apple MacPaint image files under X windows

SYNOPSIS

xmac filename [-ps] [host:display] [=geometry]

DESCRIPTION

xmac displays a MacPaint file in a window, allows resize/move, and has an icon.

xmac will send the Postscript commands to print the image to standard out if you include the command line option, *-ps*. *xmac* accepts two other optional command line arguments. You may specify a display name in the form *host:display* (see *X(1)*). And you may provide a geometry specification. If you don't give a geometry specification, *xmac* will ask you where you want to put the window when it starts up. See *X(1)* for a full explanation.

BUGS

There are no known bugs. There are lots of lacking features. I Would like to add editing capability in the future, along with the ability to clip part of an image to a bitmap format file; as well as replacing the desktop pattern with an image. Also there should be a way to kill the process, i.e. a keypress or a mouse click in a box. Also a title bar would be nice.

ENVIRONMENT

XMAC - the default directory for searching for image files, (after ".").

SEE ALSO

X(1), Xlib Documentation.

AUTHOR

Copyright (c) 1987 by Patrick J. Naughton,
(naughton@sun.soe.clarkson.edu)

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation.

NAME

`xmag` - magnify parts of the screen

SYNOPSIS

`xmag [-option ...]`

DESCRIPTION

The `xmag` program allows you to magnify portions of the screen. If no explicit region is specified, a square centered around the pointer is displayed indicating the area to be enlarged. Once a region has been selected, a window is popped up showing a blown up version of the region in which each pixel in the source image is represented by a small square of the same color. Pressing Button1 on the pointer in the enlargement window pops up a small window displaying the position, number, and RGB value of the pixel under the pointer until the button is released. Pressing the space bar or any other pointer button removes the enlarged image so that another region may be selected. Pressing “q”, “Q”, or “^C” in the enlargement window exits the program.

OPTIONS**-display *display***

This option specifies the X server to use for both reading the screen and displaying the enlarged version of the image.

-geometry *geom*

This option specifies the size and/or location of the enlargement window. By default, the size is computed from the size of the source region and the desired magnification. Therefore, only one of `-source size` and `-mag magfactor` options may be specified if a window size is given with this option.

-source *geom*

This option specifies the size and/or location of the source region on the screen. By default, a 64x64 square centered about the pointer is provided for the user to select an area of the screen. The size of the source is used with the desired magnification to compute the default enlargement window size. Therefore, only one of `-geometry size` and `-mag magfactor` options may be specified if a source size is given with this option.

-mag *magfactor*

This option specifies an integral factor by which the source region should be enlarged. The default magnification is 5. This is used with the size of the source to compute the default enlargement window size. Therefore, only one of `-geometry size` and `-source geom` options may be specified if a magnification factor is given with this option.

-bw *pixels*

This option specifies the width in pixels of the border surrounding the enlargement window.

-bd *color*

This option specifies the color to use for the border surrounding the enlargement window.

-bg *color* or *pixelvalue*

This option specifies the name of the color to be used as the background of the enlargement window. If the name begins with a percent size (%), it is interpreted to be an absolute pixel value. This is useful when displaying large areas since pixels that are the same color as the background do not need to be painted in the enlargement. The default is to use the BlackPixel of the screen.

-fn *fontname*

This option specifies the name of a font to use when displaying pixel values (used when Button1 is pressed in the enlargement window).

-z

This option indicates that the server should be grabbed during the dynamics and the call to `XGetImage`. This is useful for ensuring that clients don't change their state as a result of entering or leaving them with the pointer.

X DEFAULTS

The *xmag* program uses the following X resources:

geometry (class **Geometry**)

Specifies the size and/or location of the enlargement window.

source (class **Source**)

Specifies the size and/or location of the source region on the screen.

magnification (class **Magnification**)

Specifies the enlargement factor.

borderWidth (class **BorderWidth**)

Specifies the border width in pixels.

borderColor (class **BorderColor**)

Specifies the color of the border.

background (class **Background**)

Specifies the color or pixel value to be used for the background of the enlargement window.

font (class **Font**)

Specifies the name of the font to use when displaying pixel values when the user presses Button1 in the enlargement window.

SEE ALSO

X(1), xwd(1)

BUGS

This program will behave strangely on displays that support windows of different depths.

Because the window size equals the source size times the magnification, you only need to specify two of the three parameters. This can be confusing.

Being able to drag the pointer around and see a dynamic display would be very nice.

Another possible interface would be for the user to drag out the desired area to be enlarged.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Jim Fulton, MIT X Consortium

NAME

`xmodmap` - utility for modifying keymaps in X

SYNOPSIS

`xmodmap` [-options ...] [filename]

DESCRIPTION

The *xmodmap* program is used to edit and display the keyboard *modifier map* and *keymap table* that are used by client applications to convert event keycodes into keysyms. It is usually run from the user's session startup script to configure the keyboard according to personal tastes.

OPTIONS

The following options may be used with *xmodmap*:

- display** *display*
This option specifies the host and display to use.
- help** This option indicates that a brief description of the command line arguments should be printed on the standard error. This will be done whenever an unhandled argument is given to *xmodmap*.
- grammar**
This option indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.
- verbose**
This option indicates that *xmodmap* should print logging information as it parses its input.
- quiet** This option turns off the verbose logging. This is the default.
- n** This option indicates that *xmodmap* should not change the mappings, but should display what it would do, like *make(1)* does when given this option.
- e** *expression*
This option specifies an expression to be executed. Any number of expressions may be specified from the command line.
- pm** This option indicates that the current modifier map should be printed on the standard output.
- pk** This option indicates that the current keymap table should be printed on the standard output.
- pp** This option indicates that the current pointer map should be printed on the standard output.
- A lone dash means that the standard input should be used as the input file.

The *filename* specifies a file containing *xmodmap* expressions to be executed. This file is usually kept in the user's home directory with a name like *xmodmaprc*.

EXPRESSION GRAMMAR

The *xmodmap* program reads a list of expressions and parses them all before attempting execute any of them. This makes it possible to refer to keysyms that are being redefined in a natural way without having to worry as much about name conflicts.

keycode *NUMBER* = *KEYSYMNAME* ...

The list of keysyms is assigned to the indicated keycode (which may be specified in decimal, hex or octal and can be determined by running the *xev* program in the examples directory). Usually only one keysym is assigned to a given code.

keysym *KEYSYMNAME* = *KEYSYMNAME* ...

The *KEYSYMNAME* on the left hand side is looked up to find its current keycode and the line is replaced with the appropriate *keycode* expression. Note that if you have the same keysym bound to multiple keys, this might not work.

clear *MODIFIERNAME*

This removes all entries in the modifier map for the given modifier, where valid name are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names,

although it does matter for all other names). For example, "clear Lock" will remove all any keys that were bound to the shift lock modifier.

add *MODIFIERNAME* = *KEYSYMNAME* ...

This adds the given keysyms to the indicated modifier map. The keysym names are evaluated after all input expressions are read to make it easy to write expressions to swap keys (see the *EXAMPLES* section).

remove *MODIFIERNAME* = *KEYSYMNAME* ...

This removes the given keysyms from the indicated modifier map. Unlike **add**, the keysym names are evaluated as the line is read in. This allows you to remove keys from a modifier without having to worry about whether or not they have been reassigned.

pointer = default

This sets the pointer map back to its default settings (button 1 generates a code of 1, button 2 generates a 2, etc.).

pointer = NUMBER ...

This sets to pointer map to contain the indicated button codes. The list always starts with the first physical button.

Lines that begin with an exclamation point (!) are taken as comments.

If you want to change the binding of a modifier key, you must also remove it from the appropriate modifier map.

EXAMPLES

Many pointers are designed such the first button is pressed using the index finger of the right hand. People who are left-handed frequently find that it is more comfortable to reverse the button codes that get generated so that the primary button is pressed using the index finger of the left hand. This could be done on a 3 button pointer as follows:

```
% xmodmap -e "pointer = 3 2 1"
```

Many editor applications support the notion of Meta keys (similar to Control keys except that Meta is held down instead of Control). However, some servers do not have a Meta keysym in the default keymap table, so one needs to be added by hand. The following command will attach Meta to the Multi-language key (sometimes label Compose Character). It also takes advantage of the fact that applications that need a Meta key simply need to get the keycode and don't require the keysym to be in the first column of the keymap table. This means that applications that are looking for a Multi_key (including the default modifier map) won't notice any change.

```
% keysym Multi_key = Multi_key Meta_L
```

One of the more simple, yet convenient, uses of *xmodmap* is to set the keyboard's "rubout" key to generate an alternate keysym. This frequently involves exchanging Backspace with Delete to be more comfortable to the user. If the *tyModes* resource in *xterm* is set as well, all terminal emulator windows will use the same key for erasing characters:

```
% xmodmap -e "keysym BackSpace = Delete"
% echo "XTerm*tyModes: erase ^?" | xrdp -merge
```

Some keyboards do not automatically generate less than and greater than characters when the comma and period keys are shifted. This can be remedied with *xmodmap* by resetting the bindings for the comma and period with the following scripts:

```
!
! make shift-, be < and shift-. be >
!
keysym comma = comma less
```



```
keySYM period = period greater
```

One of the more irritating differences between keyboards is the location of the Control and Shift Lock keys. A common use of *xmodmap* is to swap these two keys as follows:

```
!
! Swap Caps_Lock and Control_L
!
remove Lock = Caps_Lock
remove Control = Control_L
keySYM Control_L = Caps_Lock
keySYM Caps_Lock = Control_L
add Lock = Caps_Lock
add Control = Control_L
```

The *keycode* command is useful for assigning the same keySYM to multiple keycodes. Although unportable, it also makes it possible to write scripts that can reset the keyboard to a known state. The following script sets the backspace key to generate Delete (as shown above), flushes all existing caps lock bindings, makes the CapsLock key be a control key, make F5 generate Escape, and makes Break/Reset be a shift lock.

```
!
! On the HP, the following keycodes have key caps as listed:
!
! 101 Backspace
! 55 Caps
! 14 Ctrl
! 15 Break/Reset
! 86 Stop
! 89 F5
!
keycode 101 = Delete
keycode 55 = Control_R
clear Lock
add Control = Control_R
keycode 89 = Escape
keycode 15 = Caps_Lock
add Lock = Caps_Lock
```

ENVIRONMENT

DISPLAY

to get default host and display number.

SEE ALSO

X(1)

BUGS

Every time a *keycode* expression is evaluated, the server generates a *MappingNotify* event on every client. This can cause some thrashing. All of the changes should be batched together and done at once. Clients that receive keyboard input and ignore *MappingNotify* events will not notice any changes made to keyboard mappings.

Xmodmap should generate "add" and "remove" expressions automatically whenever a keycode that is already bound to a modifier is changed.

There should be a way to have the *remove* expression accept keycodes as well as keysyms for those times when you really mess up your mappings.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

Copyright 1987 Sun Microsystems, Inc.

See *X(1)* for a full statement of rights and permissions.

AUTHOR

Jim Fulton, MIT X Consortium, rewritten from an original by David Rosenthal of Sun Microsystems.

NAME

xpr – print an X window dump

SYNOPSIS

xpr [*-scale scale*] [*-height inches*] [*-width inches*] [*-left inches*] [*-top inches*] [*-header string*] [*-trailer string*] [*-landscape*] [*-portrait*] [*-rv*] [*-compact*] [*-output filename*] [*-append filename*] [*-noff*] [*-split n*] [*-device dev*] [*filename*]

DESCRIPTION

Xpr takes as input a window dump file produced by *xwd(1)* and formats it for output on the LN03, LA100, PostScript printers, or IBM PP3812 page printer. If no file argument is given, the standard input is used. By default, *xpr* prints the largest possible representation of the window on the output page. Options allow the user to add headers and trailers, specify margins, adjust the scale and orientation, and append multiple window dumps to a single output file. Output is to standard output unless *-output* is specified.

Command Options

-scale scale

Affects the size of the window on the page. The LN03 and PostScript printers are able to translate each bit in a window pixel map into a grid of a specified size. For example each bit might translate into a 3x3 grid. This would be specified by *-scale 3*. By default a window is printed with the largest scale that will fit onto the page for the specified orientation.

-height inches

Specifies the maximum height of the window on the page.

-width inches

Specifies the maximum width of the window.

-left inches

Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

-top inches

Specifies the top margin for the picture in inches. Fractions are allowed.

-header header

Specifies a header string to be printed above the window.

-trailer trailer

Specifies a trailer string to be printed below the window.

-landscape

Forces the window to be printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

-portrait

Forces the window to be printed in portrait mode. By default a window is printed such that its longest side follows the long side of the paper.

-rv

Forces the window to be printed in reverse video.

-compact

Uses simple run-length encoding for compact representation of windows with lots of white pixels.

-output filename

Specifies an output file name. If this option is not specified, standard output is used.

-append filename

Specifies a filename previously produced by *xpr* to which the window is to be appended.

-noff

When specified in conjunction with *-append*, the window will appear on the same page as the

previous window.

-split *n* This option allows the user to split a window onto several pages. This might be necessary for very large windows that would otherwise cause the printer to overload and print the page in an obscure manner.

-device *device*

Specifies the device on which the file will be printed. Currently only the LN03 (-device ln03), LA100 (-device la100), PostScript printers (-device ps) and IBM PP3812 (-device pp) are supported. -device lw (LaserWriter) is equivalent to -device ps and is provided only for backwards compatibility.

SEE ALSO

xwd(1), xwud(1), X(1)

LIMITATIONS

The current version of *xpr* can generally print out on the LN03 most X windows that are not larger than two-thirds of the screen. For example, it will be able to print out a large Emacs window, but it will usually fail when trying to print out the entire screen. The LN03 has memory limitations that can cause it to incorrectly print very large or complex windows. The two most common errors encountered are "band too complex" and "page memory exceeded." In the first case, a window may have a particular six pixel row that contains too many changes (from black to white to black). This will cause the printer to drop part of the line and possibly parts of the rest of the page. The printer will flash the number '1' on its front panel when this problem occurs. A possible solution to this problem is to increase the scale of the picture, or to split the picture onto two or more pages. The second problem, "page memory exceeded," will occur if the picture contains too much black, or if the picture contains complex half-tones such as the background color of a display. When this problem occurs the printer will automatically split the picture into two or more pages. It may flash the number '5' on its front panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or to rework the application to produce a less complex picture.

Xpr provides some support for the LA100. However, there are several limitations on its use: the picture will always be printed in portrait mode, there is no scaling, and the aspect ratio will be slightly off.

Support for PostScript output currently cannot handle the **-append**, **-noff** or **-split** options.

The **-compact** option is *only* supported for PostScript output. It compresses white space but not black space, so it is not useful for reverse-video windows.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
Copyright 1986, Marvin Solomon and the University of Wisconsin.
See *X(1)* for a full statement of rights and permissions.

AUTHORS

Michael R. Gretzinger, MIT Project Athena, Jose Capo, MIT Project Athena (PP3812 support), Marvin Solomon (University of Wisconsin).

NAME

xprop - property displayer for X

SYNOPSIS

xprop [-help] [-grammar] [-id *id*] [-root] [-name *name*] [-font *font*] [-display *display*] [-remove *property-name*] [-spy] [-len *n*] [-notype] [-fs *file*] [-f *atom format [dformat]*]* [*format [dformat] atom*]*

SUMMARY

The *prop* utility is for displaying window and font properties in an X server. One window or font is selected using the command line arguments or possibly in the case of a window, by clicking on the desired window. A list of properties is then given, possibly with formatting information.

OPTIONS

- help** Print out a summary of command line options.
- grammar**
Print out a detailed grammar for all command line options.
- id *id*** This argument allows the user to select window *id* on the command line rather than using the pointer to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the pointer might be impossible or interfere with the application.
- name *name***
This argument allows the user to specify that the window named *name* is the target window on the command line rather than using the pointer to select the target window.
- font *font***
This argument allows the user to specify that the properties of font *font* should be displayed.
- root** This argument specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.
- display *display***
This argument allows you to specify the server to connect to; see *X(1)*.
- len *n*** Specifies that at most *n* bytes of any property should be read or displayed.
- notype** Specifies that the type of each property should not be displayed.
- fs *file*** Specifies that file *file* should be used as a source of more formats for properties.
- spy** Examine the window properties forever.
- remove *property-name***
Specifies the name of a property to be removed from the indicated window.
- f *name format [dformat]***
Specifies that the *format* for *name* should be *format* and that the *dformat* for *name* should be *dformat*. If *dformat* is missing, " = \$0+\n" is assumed.

DESCRIPTION

For each of these properties, its value on the selected window or font is printed using the supplied formatting information if any. If no formatting information is supplied, internal defaults are used. If a property is not defined on the selected window or font, "not defined" is printed as the value for that property. If no property list is given, all the properties possessed by the selected window or font are printed.

A window may be selected in one of four ways. First, if the desired window is the root window, the **-root** argument may be used. If the desired window is not the root window, it may be selected in two ways on the command line, either by id number such as might be obtained from *xwininfo*, or by name if the window possesses a name. The **-id** argument selects a window by id number in either decimal or hex (must start with 0x) while the **-name** argument selects a window by name.

The last way to select a window does not involve the command line at all. If none of `-font`, `-id`, `-name`, and `-root` are specified, a crosshairs cursor is displayed and the user is allowed to choose any visible window by pressing any pointer button in the desired window. If it is desired to display properties of a font as opposed to a window, the `-font` argument must be used.

Other than the above four arguments and the `-help` argument for obtaining help, and the `-grammar` argument for listing the full grammar for the command line, all the other command line arguments are used in specifying both the format of the properties to be displayed and how to display them. The `-len n` argument specifies that at most n bytes of any given property will be read and displayed. This is useful for example when displaying the cut buffer on the root window which could run to several pages if displayed in full.

Normally each property name is displayed by printing first the property name then its type (if it has one) in parentheses followed by its value. The `-notype` argument specifies that property types should not be displayed. The `-fs` argument is used to specify a file containing a list of formats for properties while the `-f` argument is used to specify the format for one property.

The formatting information for a property actually consists of two parts, a *format* and a *dformat*. The *format* specifies the actual formatting of the property (i.e., is it made up of words, bytes, or longs?, etc.) while the *dformat* specifies how the property should be displayed.

The following paragraphs describe how to construct *formats* and *dformats*. However, for the vast majority of users and uses, this should not be necessary as the built in defaults contain the *formats* and *dformats* necessary to display all the standard properties. It should only be necessary to specify *formats* and *dformats* if a new property is being dealt with or the user dislikes the standard display format. New users especially are encouraged to skip this part.

A *format* consists of one of 0, 8, 16, or 32 followed by a sequence of one or more format characters. The 0, 8, 16, or 32 specifies how many bits per field there are in the property. Zero is a special case meaning use the field size information associated with the property itself. (This is only needed for special cases like type INTEGER which is actually three different types depending on the size of the fields of the property)

A value of 8 means that the property is a sequence of bytes while a value of 16 would mean that the property is a sequence of words. The difference between these two lies in the fact that the sequence of words will be byte swapped while the sequence of bytes will not be when read by a machine of the opposite byte order of the machine that originally wrote the property. For more information on how properties are formatted and stored, consult the Xlib manual.

Once the size of the fields has been specified, it is necessary to specify the type of each field (i.e., is it an integer, a string, an atom, or what?) This is done using one format character per field. If there are more fields in the property than format characters supplied, the last character will be repeated as many times as necessary for the extra fields. The format characters and their meaning are as follows:

- a The field holds an atom number. A field of this type should be of size 32.
- b The field is an boolean. A 0 means false while anything else means true.
- c The field is an unsigned number, a cardinal.
- i The field is a signed integer.
- m The field is a set of bit flags, 1 meaning on.
- s This field and the next ones until either a 0 or the end of the property represent a sequence of bytes. This format character is only usable with a field size of 8 and is most often used to represent a string.
- x The field is a hex number (like 'c' but displayed in hex - most useful for displaying window ids and the like)

An example *format* is `32ica` which is the format for a property of three fields of 32 bits each, the first holding a signed integer, the second an unsigned integer, and the third an atom.

The format of a *dformat* unlike that of a *format* is not so rigid. The only limitations on a *dformat* is that one may not start with a letter or a dash. This is so that it can be distinguished from a property name or an argument. A *dformat* is a text string containing special characters instructing that various fields be printed at various points in a manner similar to the formatting string used by printf. For example, the *dformat* " is (\$0, \$1 \)n" would render the POINT 3, -4 which has a *format* of 32ii as " is (3, -4)n".

Any character other than a \$, ?, \, or a (in a *dformat* prints as itself. To print out one of \$, ?, \, or (precede it by a \. For example, to print out a \$, use \\$. Several special backslash sequences are provided as shortcuts. \n will cause a newline to be displayed while \t will cause a tab to be displayed. \o where o is an octal number will display character number o.

A \$ followed by a number n causes field number n to be displayed. The format of the displayed field depends on the formatting character used to describe it in the corresponding *format*. I.e., if a cardinal is described by 'c' it will print in decimal while if it is described by a 'x' it is displayed in hex.

If the field is not present in the property (this is possible with some properties), <field not available> is displayed instead. \$n+ will display field number n then a comma then field number n+1 then another comma then ... until the last field defined. If field n is not defined, nothing is displayed. This is useful for a property that is a list of values.

A ? is used to start a conditional expression, a kind of if-then statement. ?*exp(text)* will display *text* if and only if *exp* evaluates to non-zero. This is useful for two things. First, it allows fields to be displayed if and only if a flag is set. And second, it allows a value such as a state number to be displayed as a name rather than as just a number. The syntax of *exp* is as follows:

```
exp ::= term | term=exp | !exp
term ::= n | $n | mn
```

The ! operator is a logical "not", changing 0 to 1 and any non-zero value to 0. = is an equality operator. Note that internally all expressions are evaluated as 32 bit numbers so -1 is not equal to 65535. = returns 1 if the two values are equal and 0 if not. n represents the constant value n while \$n represents the value of field number n. mn is 1 if flag number n in the first field having format character 'm' in the corresponding *format* is 1, 0 otherwise.

Examples: ?m3(count: \$3\n) displays field 3 with a label of count if and only if flag number 3 (count starts at 0!) is on. ?\$2=0(True)?!\$2=0(False) displays the inverted value of field 2 as a boolean.

In order to display a property, *xprop* needs both a *format* and a *dformat*. Before *xprop* uses its default values of a *format* of 32x and a *dformat* of " = { \$0+ }n", it searches several places in an attempt to find more specific formats. First, a search is made using the name of the property. If this fails, a search is made using the type of the property. This allows type STRING to be defined with one set of formats while allowing property WM_NAME which is of type STRING to be defined with a different format. In this way, the display formats for a given type can be overridden for specific properties.

The locations searched are in order: the format if any specified with the property name (as in 8x WM_NAME), the formats defined by -f options in last to first order, the contents of the file specified by the -fs option if any, the contents of the file specified by the environmental variable XPROPFORMATS if any, and finally *xprop*'s built in file of formats.

The format of the files referred to by the -fs argument and the XPROPFORMATS variable is one or more lines of the following form:

```
name format [dformat]
```

Where *name* is either the name of a property or the name of a type, *format* is the *format* to be used with *name* and *dformat* is the *dformat* to be used with *name*. If *dformat* is not present, " = \$0+n" is assumed.

EXAMPLES

To display the name of the root window: *xprop* -root WM_NAME

To display the window manager hints for the clock: *xprop -name xclock WM_HINTS*

To display the start of the cut buffer: *xprop -root -len 100 CUT_BUFFER0*

To display the point size of the fixed font: *xprop -font fixed POINT_SIZE*

To display all the properties of window # 0x200007: *xprop -id 0x200007*

ENVIRONMENT**DISPLAY**

To get default display.

XPROPFORMATS

Specifies the name of a file from which additional formats are to be obtained.

SEE ALSO

X(1), xwininfo(1)

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xrdb - X server resource database utility

SYNOPSIS

xrdb [-option ...] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file.

The resource manager (used by the Xlib routine *XGetDefault(3X)* and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editing files.

For compatibility, if there is no RESOURCE_MANAGER property defined (either because *xrdb* was not run or if the property was removed), the resource manager will look for a file called *.Xdefaults* in your home directory.

The *filename* (or the standard input if - or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

HOST=hostname

the hostname portion of the display to which you are connected.

WIDTH=num

the width of the screen in pixels.

HEIGHT=num

the height of the screen in pixels.

X_RESOLUTION=num

the x resolution of the screen in pixels per meter.

Y_RESOLUTION=num

the y resolution of the screen in pixels per meter.

PLANES=num

the number of bit planes for the default visual.

BITS_PER_RGB=num

the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it is not related to the number of planes, which is the log base 2 of the size of the colormap.

CLASS=visualclass

one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor.

COLOR only defined if the default visual's type is one of the color options.

Lines that begin with an exclamation mark (!) are ignored and may be used as comments.

OPTIONS

xrdb program accepts the following options:

- help** This option (or any unsupported option) will cause a brief description of the allowable options and parameters to be printed.
- display *display*** This option specifies the X server to be used; see *X(1)*.
- n** This option indicates that changes to the property (when used with *-load*) or to the resource file (when used with *-edit*) should be shown on the standard output, but should not be performed.
- quiet** This option indicates that warning about duplicate entries should not be displayed.

-cpp filename

This option specifies the pathname of the C preprocessor program to be used. Although *xrdb* was designed to use CPP, any program that acts as a filter and accepts the -D, -I, and -U options may be used.

-nocpp This option indicates that *xrdb* should not run the input file through a preprocessor before loading it into the RESOURCE_MANAGER property.

-symbols

This option indicates that the symbols that are defined for the preprocessor should be printed onto the standard output. It can be used in conjunction with **-query**, but not with the options that change the RESOURCE_MANAGER property.

-query This option indicates that the current contents of the RESOURCE_MANAGER property should be printed onto the standard output. Note that since preprocessor commands in the input resource file are part of the input file, not part of the property, they won't appear in the output from this option. The **-edit** option can be used to merge the contents of the property back into the input resource file without damaging preprocessor commands.

-load This option indicates that the input should be loaded as the new value of the RESOURCE_MANAGER property, replacing whatever what there (i.e. the old contents are removed). This is the default action.

-merge This option indicates that the input should be merged with, instead of replacing, the current contents of the RESOURCE_MANAGER property. Since *xrdb* can read the standard input, this option can be used to the change the contents of the RESOURCE_MANAGER property directly from a terminal or from a shell script.

-remove

This option indicates that the RESOURCE_MANAGER property should be removed from its window.

-edit filename

This option indicates that the contents of the RESOURCE_MANAGER property should be edited into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

-backup string

This option specifies a suffix to be appended to the filename used with **-edit** to generate a backup file.

-Dname[=value]

This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef*.

-Uname This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

-Idirectory

This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include*.

FILES

Generalizes *~/.Xdefaults* files.

SEE ALSO

X(1), XGetDefault(3X), Xlib Resource Manager documentation

ENVIRONMENT**DISPLAY**

to figure out which display to use.

BUGS

The default for no arguments should be to query, not to overwrite, so that it is consistent with other programs.

COPYRIGHT

Copyright 1988, Digital Equipment Corporation.

AUTHORS

Phil Karlton, rewritten from the original by Jim Gettys

NAME

xrefresh - refresh all or part of an X screen

SYNOPSIS

xrefresh [-option ...]

DESCRIPTION

Xrefresh is a simple X program that causes all or part of your screen to be repainted. This is useful when system messages have messed up your screen. *Xrefresh* maps a window on top of the desired area of the screen and then immediately unmaps it, causing refresh events to be sent to all applications. By default, a window with no background is used, causing all applications to repaint "smoothly." However, the various options can be used to indicate that a solid background (of any color) or the root window background should be used instead.

ARGUMENTS

- white Use a white background. The screen just appears to flash quickly, and then repaint.
- black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.
- solid *color*
Use a solid background of the specified color. Try green.
- root Use the root window background.
- none This is the default. All of the windows simply repaint.
- geometry *WxH+X+Y*
Specifies the portion of the screen to be repainted; see *X(1)*.
- display *display*
This argument allows you to specify the server and screen to refresh; see *X(1)*.

X DEFAULTS

The *xrefresh* program uses the routine *XGetDefault(3X)* to read defaults, so its resource names are all capitalized.

Black, White, Solid, None, Root

Determines what sort of window background to use.

Geometry

Determines the area to refresh. Not very useful.

ENVIRONMENT

DISPLAY - To get default host and display number.

SEE ALSO

X(1)

BUGS

It should have just one default type for the background.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHORS

Jim Gettys, Digital Equipment Corp., MIT Project Athena

NAME

xset - user preference utility for X

SYNOPSIS

```
xset [-display display] [-b] [b on/off] [b [volume [pitch [duration]]] [-c] [c on/off] [c [volume]] [[-+]fp[-+=]
path[path[,...]]] [fp default] [fp rehash] [[-]led [integer]] [led on/off] [m[mouse] [acceleration [threshold]]]
[m[mouse] default] [p [pixel color] [[-]r] [r on/off] [s [length [period]]] [s blank/noblank] [s expose/noexpose]
[s on/off] [s default] [q]
```

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS**-display *display***

This option specifies the server to use; see *X(1)*.

b The **b** option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch, in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

c The **c** option controls key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, keyclick will be disabled. If a value from 0 to 100 is given, it is used to indicate volume, as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp= *path*,...

The **fp=** sets the font path to the directories given in the *path* argument. The directories are interpreted by the server, not by the client, and are server-dependent. Directories that do not contain font databases created by *mkfontdir* will be ignored by the server.

fp default

The *default* argument causes the font path to be reset to the server's default.

fp rehash

The *rehash* argument causes the server to reread the font databases in the current font path. This is generally only used when adding new fonts to a font directory (after running *mkfontdir* to recreate the font database).

-fp or fp-

The **-fp** and **fp-** options remove elements from the current font path. They must be followed by a comma-separated list of directories.

+fp or fp+

This **+fp** and **fp+** options prepend and append elements to the current font path, respectively. They must be followed by a comma-separated list of directories.

led The **led** option controls the keyboard LEDs. This controls the turning on or off of one or all of the LEDs. It accepts an optional integer, a preceding dash(-) or an 'on/off' flag. If no parameter or the 'on' flag is given, all LEDs are turned on. If a preceding dash or the flag 'off' is given, all LEDs are turned off. If a value between 1 and 32 is given, that LED will be turned on or off depending on the existence of a preceding dash. A common LED which can be controlled is the "Caps Lock" LED. "xset led 3" would turn led #3 on. "xset -led 3" would turn it off. The particular LED values may refer to different LEDs on different hardware.

- m** The **m** option controls the mouse parameters. The parameters for the mouse are 'acceleration' and 'threshold'. The mouse, or whatever pointer the machine is connected to, will go 'acceleration' times as fast when it travels more than 'threshold' pixels in a short time. This way, the mouse can be used for precise alignment when it is moved slowly, yet it can be set to travel across the screen in a flick of the wrist when desired. One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.
- p** The **p** option controls pixel color values. The parameters are the color map entry number in decimal, and a color specification. The root background colors may be changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and 1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.
- r** The **r** option controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.
- s** The **s** option lets you set the screen saver parameters. This option accepts up to two numerical parameters, a 'blank/noblank' flag, an 'expose/noexpose' flag, an 'on/off' flag, or the 'default' flag. If no parameters or the 'default' flag is used, the system will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.
- q** The **q** option gives you information on the current settings.
These settings will be reset to default values when you log out.
Note that not all X implementations are guaranteed to honor all of these options.

SEE ALSO

X(1), Xserver(1), xmodmap(1), xrdp(1), xsctroot(1)

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See X(1) for a full statement of rights and permissions.

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science
David Krikorian, MIT Project Athena (X11 version)

NAME

xsetroot – root window parameter setting utility for X

SYNOPSIS

xsetroot [-help] [-def] [-display *display*] [-cursor *cursorfile maskfile*] [-bitmap *filename*] [-mod *x y*] [-gray] [-grey] [-fg *color*] [-bg *color*] [-rv] [-solid *color*] [-name *string*]

DESCRIPTION

The *xsetroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if *-def* is specified, the window is reset to its default state. The *-def* option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (*-solid*, *-gray*, *-grey*, *-bitmap*, and *-mod*) may be specified at a time.

OPTIONS

The various options are as follows:

-help Print a usage message and exit.

-def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*

This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be made with the *bitmap(1)* program. You probably want the mask file to be all black until you get used to the way masks work.

-bitmap *filename*

Use the bitmap specified in the file to set the window pattern. You can make your own bitmap files (little pictures) using the *bitmap(1)* program. The entire background will be made up of repeated "tiles" of the bitmap.

-mod *x y*

This is used if you want a plaid-like grid pattern on your screen. *x* and *y* are integers ranging from 1 to 16. Try the different combinations. Zero and negative numbers are taken as 1.

-gray Make the entire background gray. (Easier on the eyes.)

-grey Make the entire background grey.

-fg *color*

Use "color" as the foreground color. Foreground and background colors are meaningful only in combination with *-cursor*, *-bitmap*, or *-mod*.

-bg *color*

Use "color" as the background color.

-rv This exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.

-solid *color*

Set the window color to "color".

-name *string*

Set the name of the root window to "string". There is no default value. Usually a name is assigned to a window so that the window manager can use a text representation when the window is iconified. This option is unused since you can't iconify the background.

-display *display*

Specifies the server to connect to; see *X(1)*.

SEE ALSO

X(1), xset(1), xrdb(1)

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

`xterm` – terminal emulator for X

SYNOPSIS

`xterm [-toolkitoption ...] [-option ...]`

DESCRIPTION

The `xterm` program is a terminal emulator for the X Window System. It provides DEC VT102 and Tektronix 4014 compatible terminals for programs that can't use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), `xterm` will use the facilities to notify programs running in the window whenever it is resized.

The VT102 and Tektronix 4014 terminals each have their own window so that you can edit text in one and look at graphics in the other at the same time. To maintain the correct aspect ratio (height/width), Tektronix graphics will be restricted to the largest box with a 4014's aspect ratio that will fit in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the "active" window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be chosen through escape sequences, the "Modes" menu in the VT102 window, and the "Tektronix" menu in the 4014 window.

OPTIONS

The `xterm` terminal emulator accepts all of the standard X Toolkit command line options along with the additional options listed below (if the option begins with a '+' instead of a '-', the option is restored to its default value):

- help** This causes `xterm` to print out a verbose message describing its options.
- 132** Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored. This option causes the DECCOLM escape sequence to be recognized, and the `xterm` window will resize appropriately.
- ah** This option indicates that `xterm` should always highlight the text cursor and borders. By default, `xterm` will display a hollow text cursor whenever the focus is lost or the pointer leaves the window.
- +ah** This option indicates that `xterm` should do text cursor highlighting.
- b number**
This option specifies the size of the inner border (the distance between the outer edge of the characters and the window border) in pixels. The default is 2.
- cc characterclassrange:value[,...]**
This sets classes indicated by the given ranges for using in selecting by words. See the section specifying character classes.
- cr color**
This option specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.
- cu** This option indicates that `xterm` should work around a bug in the `curses(3x)` cursor motion package that causes the `more(1)` program to display lines that are exactly the width of the window and are followed by line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).
- +cu** This option indicates that that `xterm` should not work around the `curses(3x)` bug mentioned above.
- e program [arguments ...]**
This option specifies the program (and its command line arguments) to be run in the `xterm`

window. It also sets the window title and icon name to be the basename of the program being executed if neither *-T* nor *-n* are given on the command line. **This must be the last option on the command line.**

- fb font** This option specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default bold font is "vtbold."
- j** This option indicates that *xterm* should do jump scrolling. Normally, text is scrolled one line at a time; this option allows *xterm* to move multiple lines at a time so that it doesn't fall as far behind. Its use is strongly recommended since it make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.
- +j** This option indicates that *xterm* should not do jump scrolling.
- l** This option indicates that *xterm* should send all terminal output to a log file as well as to the screen. This option can be enabled or disabled using the "xterm X11" menu.
- +l** This option indicates that *xterm* should not do logging.
- lf filename**
This option specifies the name of the file to which the output log described above is written. If *file* begins with a pipe symbol (*|*), the rest of the string is assumed to be a command to be used as the endpoint of a pipe. The default filename is "XtermLog.XXXXX" (where XXXXX is the process id of *xterm*) and is created in the directory from which *xterm* was started (or the user's home directory in the case of a login window).
- ls** This option indicates that shell that is started in the *xterm* window be a login shell (i.e. the first character of *argv[0]* will be a dash, indicating to the shell that it should read the user's .login or .profile).
- +ls** This option indicates that the shell that is started should not be a login shell (i.e. it will be normal "subshell").
- mb** This option indicates that *xterm* should ring a margin bell when the user types near the right end of a line. This option can be turned on and off from the "Modes" menu.
- +mb** This option indicates that margin bell should not be rung.
- ms color**
This option specifies the color to be used for the pointer cursor. The default is to use the foreground color.
- nb number**
This option specifies the number of characters from the right end of a line at which the margin bell, if enabled, will ring. The default is 10.
- rw** This option indicates that reverse-wraparound should be allowed. This allows the cursor to back up from the leftmost column of one line to the rightmost column of the previous line. This is very useful for editing long shell command lines and is encouraged. This option can be turned on and off from the "Modes" menu.
- +rw** This option indicates that reverse-wraparound should not be allowed.
- s** This option indicates that *xterm* may scroll asynchronously, meaning that the screen does not have to be kept completely up to date while scrolling. This allows *xterm* to run faster when network latencies are very high and is typically useful when running across a very large internet or many gateways.
- +s** This option indicates that *xterm* should scroll synchronously.
- sb** This option indicates that some number of lines that are scrolled off the top of the window should

be saved and that a scrollbar should be displayed so that those lines can be viewed. This option may be turned on and off from the "Modes" menu.

- +sb** This option indicates that a scrollbar should not be displayed.
- sf** This option indicates that Sun Function Key escape codes should be generated for function keys.
- +sf** This option indicates that the standard escape codes should be generated for function keys.
- si** This option indicates that output to a window should not automatically reposition the screen to the bottom of the scrolling region. This option can be turned on and off from the "Modes" menu.
- +si** This option indicates that output to a window should cause it to scroll to the bottom.
- sk** This option indicates that pressing a key while using the scrollbar to review previous lines of text should cause the window to be repositioned automatically in the normal position at the bottom of the scroll region.
- +sk** This option indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.
- sl *number***
This option specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.
- t** This option indicates that *xterm* should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the "Modes" menus.
- +t** This option indicates that *xterm* should start in VT102 mode.
- vb** This option indicates that a visual bell is preferred over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.
- +vb** This option indicates that a visual bell should not be used.
- C** This option indicates that this window should be receive console output. This is not supported on all systems.
- L** This option indicates that *xterm* was started by *init*. In this mode, *xterm* does not try to allocate a new pseudoterminal as *init* has already done so. In addition, the system program *getty* is run instead of the user's shell. **This option should never be used by users when starting terminal windows.**
- Sccn** This option specifies the last two letters of the name of a pseudoterminal to use in slave mode. This allows *xterm* to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

- %geom** This option specifies the preferred size and position of the Tektronix window. It is shorthand for specifying the "**tekGeometry*" resource.
- #geom** This option specifies the preferred position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.
- T *string***
This option specifies the title for *xterm*'s windows. It is equivalent to **-title**.
- nstring** This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource.
- r** This option indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.
- w *number***

This option specifies the width in pixels of the border surrounding the window. It is equivalent to **-borderwidth** or **-bw**.

The following standard X Toolkit command line arguments are commonly used with *xterm*:

-bg color

This option specifies the color to use for the background of the window. The default is "white."

-bd color

This option specifies the color to use for the border of the window. The default is "black."

-bw number

This option specifies the width in pixels of the border surrounding the window.

-fg color

This option specifies the color to use for displaying text. The default is "black".

-fn font This option specifies the font to be used for displaying normal text. The default is "vtsingle."

-name name

This option specifies the application name under which resource are to be obtained, rather than the default executable file name.

-rv

This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-geometry geometry

This option specifies the preferred size and position of the VT102 window; see *X(1)*;

-display display

This option specifies the X server to contact; see *X(1)*.

-xrm resourcestring

This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

-iconic

This option indicates that *xterm* should ask the window manager to start it as an icon rather than as the normal window.

X DEFAULTS

The program understands all of the core X Toolkit resource names and classes as well as:

name (class Name)

Specifies the name of this instance of the program. The default is "xterm."

iconGeometry (class IconGeometry)

Specifies the preferred size and position of the application when iconified. It is not necessarily obeyed by all window managers.

title (class Title)

Specifies a string that may be used by the window manager when displaying this application.

utmpInhibit (class UtmpInhibit)

Specifies whether or not *xterm* should try to record the user's terminal in */etc/utmp*.

sunFunctionKeys (class SunFunctionKeys)

Specifies whether or not Sun Function Key escape codes should be generated for function keys instead of standard escape sequences.

The following resources are specified as part of the "vt100" widget (class "VT100"):

alwaysHighlight (class AlwaysHighlight)

Specifies whether or not *xterm* should always display a highlighted text cursor. By default, a hollow text cursor is displayed whenever the pointer moves out of the window or the window loses the input focus.

font (class **Font**)

Specifies the name of the normal font. The default is "vtsingle."

boldFont (class **Font**)

Specifies the name of the bold font. The default is "vtbold."

c132 (class **C132**)

Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

charClass (class **CharClass**)

Specifies comma-separated lists of character class bindings of the form *[low-]high:value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

courses (class **Courses**)

Specifies whether or not the last column bug in cursors should be worked around. The default is "false."

background (class **Background**)

Specifies the color to use for the background of the window. The default is "white."

foreground (class **Foreground**)

Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the "text" color change color. The default is "black."

cursorColor (class **Foreground**)

Specifies the color to use for the text cursor. The default is "black."

geometry (class **Geometry**)

Specifies the preferred size and position of the VT102 window.

tekGeometry (class **Geometry**)

Specifies the preferred size and position of the Tektronix window.

internalBorder (class **BorderWidth**)

Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class **JumpScroll**)

Specifies whether or not jump scroll should be used. The default is "false".

logFile (class **Logfile**)

Specifies the name of the file to which a terminal session is logged. The default is "XtermLog.XXXXX" (where XXXXX is the process id of *xterm*).

logging (class **Logging**)

Specifies whether or not a terminal session should be logged. The default is "false."

logInhibit (class **LogInhibit**)

Specifies whether or not terminal session logging should be inhibited. The default is "false."

loginShell (class **LoginShell**)

Specifies whether or not the shell to be run in the window should be started as a login shell. The default is "false."

marginBell (class **MarginBell**)

Specifies whether or not the bell should be run when the user types near the right margin. The default is "false."

multiScroll (class **MultiScroll**)

Specifies whether or not asynchronous scrolling is allowed. The default is "false."

nMarginBell (class **Column**)

Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.

pointerColor (class **Foreground**)

Specifies the color of the pointer. The default is "black."

pointerShape (class **Cursor**)

Specifies the name of the shape of the pointer. The default is "xterm."

reverseVideo (class **ReverseVideo**)

Specifies whether or not reverse video should be simulated. The default is "false."

reverseWrap (class **ReverseWrap**)

Specifies whether or not reverse-wraparound should be enabled. The default is "false."

saveLines (class **SaveLines**)

Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class **ScrollBar**)

Specifies whether or not the scrollbar should be displayed. The default is "false."

scrollInput (class **ScrollCond**)

Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "true."

scrollKey (class **ScrollCond**)

Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is "false."

signalInhibit (class **SignalInhibit**)

Specifies whether or not the entries in the "xterm X11" menu for sending signals to *xterm* should be disallowed. The default is "false."

tekInhibit (class **TekInhibit**)

Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekStartup (class **TekStartup**)

Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class **TiteInhibit**)

Specifies whether or not *xterm* should remove *ti* or *te* termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string.

visualBell (class **VisualBell**)

Specifies whether or not a visible bell (i.e. flashing) should be used instead of an audible bell when Control-G is received. The default is "false."

The following resources are specified as part of the "tek4014" widget (class "Tek4014"):

width (class **Width**)

Specifies the width of the Tektronix window in pixels.

height (class **Height**)

Specifies the height of the Tektronix window in pixels.

The following resources are specified as part of the "menu" widget:

menuBorder (class **MenuBar**)

Specifies the size in pixels of the border surrounding menus. The default is 2.

menuFont (class **Font**)

Specifies the name of the font to use for displaying menu items.

menuPad (class **MenuPad**)

Specifies the number of pixels between menu items and the menu border. The default is 3.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness**)

Specifies the width in pixels of the scrollbar.

background (class **Background**)

Specifies the color to use for the background of the scrollbar.

foreground (class **Foreground**)

Specifies the color to use for the foreground of the scrollbar. The “thumb” of the scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap(5)* entries that work with *xterm* include “*xterm*”, “*vt102*”, “*vt100*” and “*ansi*”, and *xterm* automatically searches the termcap file in this order for these entries and then sets the “*TERM*” and the “*TERMCAP*” environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the “*Xterm Control Sequences*” document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the **Tektronix** menu; see below). The name of the file will be “*COPYyy-MM-dd.hh:mm:ss*”, where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login *xterm*).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the “shift” key.

Pointer button one (usually left) is used to save text into the cut buffer. Move the cursor to beginning of the text, and then hold the button down while moving the cursor to the end of the region and releasing the button. The selected text is highlighted and is saved in the global cut buffer when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection.

Pointer button two (usually middle) ‘types’ (pastes) the text from the cut buffer, inserting it as keyboard input.

Pointer button three (usually right) extends the current selection. (Without loss of generality, that is you can swap “right” and “left” everywhere in the rest of this paragraph...) If pressed while closer to the right edge of the selection than the left, it extends/contracts the right edge of the selection. If you contract the selection past the left edge of the selection, *xterm* assumes you really meant the left edge, restores the original selection, then extends/contracts the left edge of the selection. Extension starts in the selection unit mode that the last selection or extension was performed in; you can multiple-click to cycle through them.

By cutting and pasting pieces of text without trailing new lines, you can take text from several places in different windows and form a command to the shell, for example, or take output from a program and insert it into your favorite editor. Since the cut buffer is globally shared among different applications, you should

regard it as a 'file' whose contents you know. The terminal emulator and other text programs should be treating it as if it were a text file, i.e. the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking button three moves the top line of the display window down to the pointer position.

Clicking button two moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window does not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing button one, two, or three will return the letters 'l', 'm', and 'r', respectively. If the 'shift' key is pressed when a pointer button is pressed, the corresponding upper case letter is sent. To distinguish a pointer button from a key, the high bit of the character is set (but this bit is normally stripped unless the terminal mode is RAW; see *tty(4)* for details).

MENUS

Xterm has three different menus, named **xterm**, **Modes**, and **Tektronix**. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two sections, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The modes section contains items that apply to both the VT102 and Tektronix windows. Notable entries in the command section of the menu are the **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate** and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM and SIGKILL signals, respectively, to the process group of the process running under *xterm* (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The **Modes** menu sets various modes in the VT102 emulation, and is popped up when the "control" key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to their initial states just after *xterm* has finished processing the command line options. The **Tektronix** menu sets various modes in the Tektronix emulation, and is popped up when the "control" key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

CHARACTER CLASSES

Clicking the middle mouse button twice in rapid succession will cause all characters of the same class (e.g. letters, white space, punctuation) to be selected. Since different people have different preferences for what should be selected (for example, should filenames be selected as a whole or only the separate subnames), the default mapping can be overridden through the use of the *clarClass* (class *CharClass*) resource.

This resource is simply a list of *range:value* pairs where the range is either a single number or *low-high* in the range of 0 to 127, corresponding to the ASCII code for the character or characters to be set. The *value* is arbitrary, although the default table uses the character number of the first character occurring in the set.

The default table is:

```
static int charClass[128] = {
/* NUL SOH STX ETX EOT ENQ ACK BEL */
```

```

32, 1, 1, 1, 1, 1, 1, 1,
/* BS HT NL VT NP CR SO SI */
1, 32, 1, 1, 1, 1, 1, 1,
/* DLE DC1 DC2 DC3 DC4 NAK SYN ETB */
1, 1, 1, 1, 1, 1, 1, 1,
/* CAN EM SUB ESC FS GS RS US */
1, 1, 1, 1, 1, 1, 1, 1,
/* SP ! " # $ % & ' */
32, 33, 34, 35, 36, 37, 38, 39,
/* ( ) * + , - . / */
40, 41, 42, 43, 44, 45, 46, 47,
/* 0 1 2 3 4 5 6 7 */
48, 48, 48, 48, 48, 48, 48, 48,
/* 8 9 : ; < = > ? */
48, 48, 58, 59, 60, 61, 62, 63,
/* @ A B C D E F G */
64, 48, 48, 48, 48, 48, 48, 48,
/* H I J K L M N O */
48, 48, 48, 48, 48, 48, 48, 48,
/* P Q R S T U V W */
48, 48, 48, 48, 48, 48, 48, 48,
/* X Y Z [ \ ] ^ _ */
48, 48, 48, 91, 92, 93, 94, 48,
/* ` a b c d e f g */
96, 48, 48, 48, 48, 48, 48, 48,
/* h i j k l m n o */
48, 48, 48, 48, 48, 48, 48, 48,
/* p q r s t u v w */
48, 48, 48, 48, 48, 48, 48, 48,
/* x y z { | } ~ DEL */
48, 48, 48, 123, 124, 125, 126, 1);

```

For example, the string “33:48,37:48,45-47:48,64:48” indicates that the exclamation mark, percent sign, dash, period, slash, and ampersand characters should be treated the same way as characters and numbers. This is very useful for cutting and pasting electronic mailing addresses and Unix filenames.

KEY TRANSLATIONS

It is possible to rebind keys (or sequences of keys) to arbitrary strings for input, by changing the translations for the vt100 or tek4014 widgets. Changing the translations for events other than key presses is not expected, and will cause unpredictable behavior.

The actions available for key translations are “insert”, “string”, “keymap”, and “selection”. The insert action causes the key to be processed in the normal way. The string action takes a single string argument, and rebinds the key/sequence to that string value. The keymap action takes a single string argument naming a resource to be used to dynamically define a new translation table; the name of the resource is obtained by appending the string “Keymap” to the string argument to the action. A string argument of “None” to the keymap action restores the original translation table (the very first one; a stack is not maintained). The selection action takes a single string argument specifying a selection and causes the contents of that selection to be used as effective input (the current implementation of this is a crock, and always uses cut buffer zero).

For example, helpers for dbx on a Sun:

```

*VT100.translations: #override <Key>L7: keymap(dbx)
*VT100.dbxKeymap.translations: \

```

```

<Key>L7: keymap(None)          \n\
<Key>L9: string("next") string(0x0d) \n\
<Key>L10: string("step") string(0x0d) \n\
<Key>L8: string("print ") selection(PRIMARY)

```

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replaced with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

SEE ALSO

resize(1), X(1), pty(4), tty(4)
 "Xterm Control Sequences"

BUGS

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the pty and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after *xterm* starts.

This manual page is too long. There should be a separate users manual defining all of the non-standard escape sequences.

All programs should be written to use X directly; then we could eliminate this program.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
 See *X(1)* for a full statement of rights and permissions.

AUTHORS

Far too many people, including:

Loretta Guarino Reid (DEC-UEG-WSL), Joel McCormack (DEC-UEG-WSL), Terry Weissman (DEC-UEG-WSL), Edward Moy (Berkeley), Ralph R. Swick (MIT-Athena), Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT X Consortium), Doug Mink (SAO), Steve Pitschke (Stellar), Ron Newman (MIT-Athena), Jim Fulton (MIT X Consortium)

NAME

`xwd` - dump an image of an X window

SYNOPSIS

`xwd` [-debug] [-help] [-nobdrs] [-out *file*] [-xy] [-frame] [-display *display*]

DESCRIPTION

Xwd is an X Window System window dumping utility. *Xwd* allows X users to store window images in a specially formatted dump file. This file can then be read by various other X utilities for redisplay, printing, editing, formatting, archiving, image processing, etc. The target window is selected by clicking the mouse in the desired window. The keyboard bell is rung once at the beginning of the dump and twice when the dump is completed.

OPTIONS**-display *display***

This argument allows you to specify the server to connect to; see *X(1)*.

-help Print out the 'Usagc:' command syntax summary.

-nobdrs This argument specifies that the window dump should not include the pixels that compose the X window border. This is useful in situations where you may wish to include the window contents in a document as an illustration.

-out *file* This argument allows the user to explicitly specify the output file on the command line. The default is to output to standard out.

-xy This option applies to color displays only. It selects 'XY' format dumping instead of the default 'Z' format.

-add *value*

This option specifies an signed value to be added to every pixel.

-frame This option indicates that the window manager frame should be included when manually selecting a window.

ENVIRONMENT**DISPLAY**

To get default host and display number.

FILES**XWDFile.h**

X Window Dump File format definition file.

SEE ALSO

xwud(1), *xpr(1)*, *X(1)*

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

See *X(1)* for a full statement of rights and permissions.

AUTHORS

Tony Della Fera, Digital Equipment Corp., MIT Project Athena

William F. Wyatt, Smithsonian Astrophysical Observatory

NAME

`xwininfo` - window information utility for X

SYNOPSIS

`xwininfo` [-help] [-id *id*] [-root] [-name *name*] [-int] [-tree] [-stats] [-bits] [-events] [-size] [-wm] [-all] [-english] [-metric] [-display *display*]

DESCRIPTION

`Xwininfo` is a utility for displaying information about windows. Various information is displayed depending on which options are selected. If no options are chosen, `-stats` is assumed.

The user has the option of selecting the target window with the mouse (by clicking any mouse button in the desired window) or by specifying its window id on the command line with the `-id` option. Or instead of specifying the window by its id number, the `-name` option may be used to specify which window is desired by name. There is also a special `-root` option to quickly obtain information on X's root window.

OPTIONS

- help** Print out the 'Usage:' command syntax summary.
- id *id*** This option allows the user to specify a target window *id* on the command line rather than using the mouse to select the target window. This is very useful in debugging X applications where the target window is not mapped to the screen or where the use of the mouse might be impossible or interfere with the application.
- name *name*** This option allows the user to specify that the window named *name* is the target window on the command line rather than using the mouse to select the target window.
- root** This option specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.
- int** This option specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.
- tree** This option causes the root, parent, and children windows' ids and names of the selected window to be displayed.
- stats** This option causes the display of various attributes pertaining to the location and appearance of the selected window. Information displayed includes the location of the window, its width and height, its depth, border width, class, colormap id if any, map state, backing-store hint, and location of the corners.
- bits** This option causes the display of various attributes pertaining to the selected window's raw bits and how the selected window is to be stored. Displayed information includes the selected window's bit gravity, window gravity, backing-store hint, backing-planes value, backing pixel, and whether or not the window has save-under set.
- events** This option causes the selected window's event masks to be displayed. Both the event mask of events wanted by some client and the event mask of events not to propagate are displayed.
- size** This option causes the selected window's sizing hints to be displayed. Displayed information includes: for both the normal size hints and the zoom size hints, the user supplied location if any; the program supplied location if any; the user supplied size if any; the program supplied size if any; the minimum size if any; the maximum size if any; the resize increments if any; and the minimum and maximum aspect ratios if any.
- wm** This option causes the selected window's window manager hints to be displayed. Information displayed may include whether or not the application accepts input, what the window's icon window # and name is, where the window's icon should go, and what the window's initial state should be.

- metric** This option causes all individual height, width, and x and y positions to be displayed in millimeters as well as number of pixels, based on what the server thinks the resolution is. Geometry specifications that are in +x+y form are not changed.
- english** This option causes all individual height, width, and x and y positions to be displayed in inches (and feet, yards, and miles if necessary) as well as number of pixels. **-metric** and **-english** may both be enabled at the same time.
- all** This option is a quick way to ask for all information possible.
- display *display***
This option allows you to specify the server to connect to; see *X(1)*.

EXAMPLE

The following is a sample summary taken with no options specified:

```
xwininfo ==> Please select the window about which you
           ==> would like information by clicking the
           ==> mouse in that window.
```

```
xwininfo ==> Window id: 0x60000f (xterm)
```

```
           ==> Upper left X: 4
           ==> Upper left Y: 19
           ==> Width: 726
           ==> Height: 966
           ==> Depth: 4
           ==> Border width: 3
           ==> Window class: InputOutput
           ==> Colormap: 0x80065
           ==> Window Bit Gravity State: NorthWestGravity
           ==> Window Window Gravity State: NorthWestGravity
           ==> Window Backing Store State: NotUseful
           ==> Window Save Under State: no
           ==> Window Map State: IsViewable
           ==> Window Override Redirect State: no
           ==> Corners: +4+19 -640+19 -640-33 +4-33
```

ENVIRONMENT**DISPLAY**

To get the default host and display number.

SEE ALSO

X(1), *xprop(1)*

BUGS

Using **-stats -bits** shows some redundant information.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xwud - image displayer for X

SYNOPSIS

xwud [-debug] [-help] [-inverse] [-in *file*] [-display *display*]

DESCRIPTION

Xwud is an X Window System window image undumping utility. *Xwud* allows X users to display window images that were saved by *xwd* in a specially formatted dump file. The window image will appear at the coordinates of the original window from which the dump was taken. This is a crude version of a more advanced utility that has never been written. Monochrome dump files are displayed on a color monitor in the default foreground and background colors.

OPTIONS

- help** Print out a short description of the allowable options.
- in *file*** This option allows the user to explicitly specify the input file on the command line. If no input file is given, the standard input is assumed.
- inverse** Applies to monochrome window dump files only. If selected, the window is undumped in reverse video. This is mainly needed because the display is 'write white', whereas dump files intended eventually to be written to a printer are generally 'write black'.
- display *display***
This option allows you to specify the server to connect to; see *X(1)*.

ENVIRONMENT**DISPLAY**

To get default display.

FILES**XWDFFile.h**

X Window Dump File format definition file.

BUGS

Does not attempt to do color translation when the destination screen does not have a colormap exactly matching that of the original window.

SEE ALSO

xwd(1), *xpr(1)*, *X(1)*

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.
See *X(1)* for a full statement of rights and permissions.

AUTHOR

Tony Della Fera, Digital Equipment Corp., MIT Project Athena
William F. Wyatt, Smithsonian Astrophysical Observatory

Index

A

abbreviating commands, 14
access list
 modifying, 13
accessing and scaling fonts, 49
adding fonts, 51

B

bitmap fonts, 45
bitmap(n), 135
bldfamily(1), 51, 69
buildmenu(1), 13, 71

C

changing color of root canvas, 12, 15
changing the initial screen image, 11
ClassBaseFrame, 35
/ClassCanvas, 35
client/server model, 3
clients, 3, 5
color, 57
 colormaps, 57
 dynamic colors, 60
 specifying, 57
 support in X11/News, 57
 visuals, 57
colormaps
 dynamic, 58
 static, 58
compiling source programs, 21
ConfigureWindow, 37
connecting to remote X11/News servers, 19
conserving dictionary space, 12
convertfont(1), 51, 73
core set of outline fonts, 46
CPS facility, 27
 cps(1), 75
 psh(1), 107
cps(1), 75
creating your own dictionaries, 12

D

dictionary
 system, 12
dictionary space

dictionary space, *continued*
 conserving, 12
DISPLAY, 39
dumpfont, see *convertfont*
dynamic colormaps, 58

E

executive, 18
external window management, 35, 40

F

F3 format fonts, 45
font families, 49
FONTPATH, 45
fonts, 45
 bldfamily(1), 69
 convertfont(1), 73
 dumpfont, see *convertfont*
 mkiconfont(1), 85
formatting code, 109
frame object, 35

G

get, 13

I

ico(n), 143
init.ps, 4, 10
initialization files, 10
InitPaintRoot
 redefining, 12
interactive
 server, 18
interactive News programming, 18

J

journalling(1), 77

K

kbd_mode(1), 79
 News and SunView1, 64
kernel-based window system, 5
keyboard
 resetting input, 79

L

ldf(1), 81
logo(n), 145

M

makeafb(1), 51, 83
MapRequest, 37
maze(n), 147
minimum set of fonts, 49
mkiconfont(1), 85
modifying startup.ps, 11
modifying the access list, 13
modifying the root menu, 13
modifying the X11/News server, 9
muncher(n), 149

N

namestripe, 35
/new
 in X resource class, 28
News window system, 3, 9
 I/O library functions, 113
 initialization files, 10
 interactive programming, 18
 running programs, 17
 type extensions, 9
 utilities, 9 thru 10
 window management, 35
News applications, 5
News toolkit, 4, 35, 38
newshost(1), 89
NEWSSERVER, 19, 39
newsserverstr(1), 91

O

objectdiff(1), 93
objectwatcher(1), 95
on(1), 20, 30
OPEN LOOK, 4
OpenFonts, 45
openwin-init, 4, 39
openwindemos(6), 97
OpenWindows user environment, 4
outline fonts, 45
override-redirect attribute, 38, 40

P

pageview(1), 18, 103
/PaintCanvas, 35
pam(1), 105
pixel-based imaging model, 3
plaid(n), 151
POSTSCRIPT language, 3, 9
 language files, 9, 17
 previewing graphics, 18
previewing POSTSCRIPT graphics, 18
*.ps files
 init.ps, 10
psh(1), 17, 18, 107

psindent(1), 109
psic(3), 111
psman(1), 117
psps(1), 119
pstags(1), 121
psterm(1), 123
pswm, 38, 39
pswm(1), 153
put, 13
puzzle(n), 155

R

root canvas
 changing color, 12, 15
root menu
 modifying, 13
running News programs, 17
running a News-only server, 21
running an X11-only server, 30, 39
running remote X11/News clients, 20, 30
running SunView applications, 63

S

server-based window system, 5
server-supplied fonts, 45
source programs
 compiling, 21
 .startup.ps, 4, 11
 modifying, 11
static colormaps, 58
stencil/paint imaging model, 3
store, 13
SubstructureRedirect, 37, 38
SunView, 3, 5
 bugs in using, 64
 overlapping X11/News windows, 63
 running applications, 63
 using with X11/News, 63
SunView menu files, 13
SunWindows, 5, 35

T

type extensions in News, 9

U

.user.ps, 4, 11
 modifying, 12
UserProfile dictionary, 15

V

visuals, 57
 color mappings, 58
 default visual, 58
 PseudoColor, 58
 StaticColor, 58

W

window, 5
window management, 35, 5

window manager, 35
 window server, 5
 worm(n), 157

xwininfo(n), 211
 xwm, 39
 xwud(n), 213

X

X resource class

/new, 28
 /XConstructID, 28
 /XLookupClient, 28
 /XLookupID, 28

X11

BlackPixel(), 31
 displaying POSTSCRIPT images, 26
 features not supported, 26
 GXclear, 31
 GXset, 31
 idioms to avoid, 31
 redirection, 36
 utilities, 25
 WhitePixel(), 31
 window management, 35, 36, 39

X11 applications, 4, 5

X11 window system, 3

X11/News

connecting to remote servers, 19
 interactive programming, 18
 restricted access, 30
 running a News-only server, 21
 running an X11-only server, 30, 39
 running remote clients, 20, 30
 window management, 38

X11/News server, 3

modifications, 9
 start-up procedure, 4

xcalc(n), 159

/XConstructID

in X resource class, 28

xdpr(n), 163

xdpyinfo(n), 165

xfd(n), 167

xhost(n), 169

/XLookupClient

in X resource class, 28

/XLookupID

in X resource class, 28

xlsfonts(n), 171

xlswins(n), 173

xmac(n), 175

xmag(n), 177

xmodmap(n), 179

xnews(1), 4, 127

xpr(n), 183

xprop(n), 185

xrdb(n), 189

xrefresh(n), 193

xset(n), 195

xsetroot(n), 197

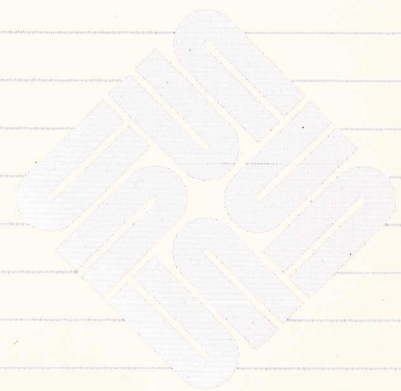
xterm(n), 199

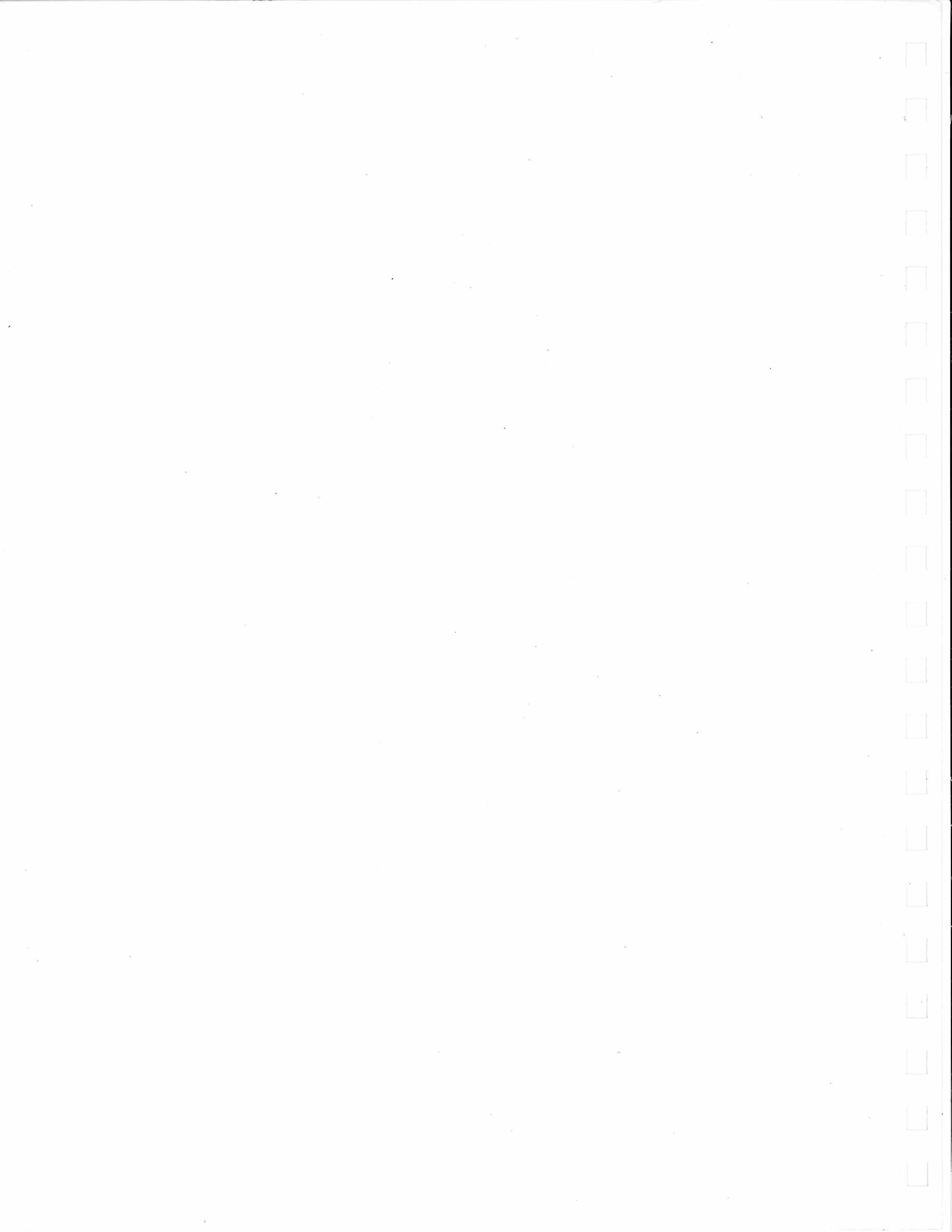
XView toolkit, 4

xwd(n), 209



X11/NeWS 1.0 Server Guide





Systems for Open Computing™

Corporate Headquarters
Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
415 960-1300
FAX 415 969-9131

**For U.S. Sales Office
locations, call:**
800 821-4643
In CA: 800 821-4642

European Headquarters
Sun Microsystems Europe, Inc.
Bagshot Manor, Green Lane
Bagshot, Surrey GU19 5NL
England
0276 51440
TLX 859017

Australia: (02) 413 2666
Canada: 416 477-6745
France: (1) 40 94 80 00

Germany: (089) 95094-0
Hong Kong: 852 5-8651688
Italy: (39) 6056337
Japan: (03) 221-7021
Korea: 2-7802255
New Zealand: (04) 499 2344
Nordic Countries: +46 (0)8 7647810
PRC: 1-8315568
Singapore: 224 3388
Spain: (1) 2532003
Switzerland: (1) 8289555
The Netherlands: 033 501234

Taiwan: 2-7213257
UK: 0276 62111

**Europe, Middle East, and Africa,
call European Headquarters:**
0276 51440

**Elsewhere in the world,
call Corporate Headquarters:**
415 960-1300
Intercontinental Sales

