CONVEX CXwindows Programmer's Reference

Document No. 710-004530-002

Second Edition
June 1990

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX CXwindows Programmer's Reference
Order No. DSW-233
Second Edition

Printed in the United States of America

Revision Information for
CONVEX CXwindows Programmer's Reference

Edition	Document No.	Description
Second	710-004530-002	Released with CXwindows V2.0 in June 1990. Includes new X clients. Refer to the Abstract.
First	710-004530-001	Initially released with CXwindows V1.1 in June 1989.

CXwindows 2.0

Optional Product

CXwindows provides a common graphics interface to link a variety of workstations and X terminals to CONVEX supercomputers. It gives users and programmers a flexible window environment that can be displayed on most bit-mapped systems, has a transparent interface, makes applications available to an entire network without reprogramming, and provides a stable programming library.

CONVEX CXwindows documentation:

CONVEX CXwindows Programmer's Reference
CONVEX OSF/Motif and CXwindows User's Guide
CONVEX CXwindows Release Notice
CONVEX CXwindows Installation Procedures

O'Reilly Xlib Programming Manual I
O'Reilly Xlib Reference Manual II
O'Reilly X Window System User's Guide III
O'Reilly X Intrinsics Programming Manual IV
O'Reilly X Intrinsics Reference Manual V

List of Entries:

XCirculateEvent(3)
XCirculateRequestEvent(3)
XClearArea(3)
XClientMessageEvent(3)
XColormapEvent(3)
XConfigureEvent(3)
XConfigureRequestEvent(3)
XConfigureWindow(3)
XConsortium(1)
XCopyArea(3)
XCreateColormap(3)
XCreateFontCursor(3)
XCreateGC(3)
XCreateImage(3)
XCreatePixmap(3)
XCreateRegion(3)
XCreateWindow(3)
XCreateWindowEvent(3)
XCrossingEvent(3)
XDefineCursor(3)
XDestroyWindow(3)
XDestroyWindowEvent(3)
XDrawArc(3)
XDrawImageString(3)
XDrawLine(3)
XDrawPoint(3)
XDrawRectangle(3)
XDrawString(3)
XDrawText(3)
XEmptyRegion(3)
XErrorEvent(3)
XExposeEvent(3)
XFillRectangle(3)
XFlush(3)
XFocusChangeEvent(3)
XFree(3)
XGetDefault(3)
XGetVisualInfo(3)
XGetWindowAttributes(3)
XGetWindowProperty(3)
XGrabButton(3)
XGrabKey(3)
XGrabKeyboard(3)
XGrabPointer(3)
XGrabServer(3)
XGraphicsExposeEvent(3)
XSetCloseDownMode(3)
XSetCommand(3)
XSetErrorHandler(3)
XSetFillStyle(3)
XSetFont(3)
XSetFontPath(3)
XSetInputFocus(3)
XSetLineAttribute(3)
XSetPointerMapping(3)
XSetScreenSaver(3)
XSetSelectionOwner(3)
XSetState(3)
XSetTextProperty(3)
XSetTile(3)
XSetTransientForHint(3)
XSetWMClientMachine(3)
XSetWMColormapWindows(3)
XSetWMIconName(3)
XSetWMName(3)
XSetWMProperties(3)
XSetWMProtocols(3)
XStandards(1)
XStoreBytes(3)
XStoreColors(3)
XStringListToTextProperty(3)
XStringToKeysym(3)
XSynchronize(3)
XTextExtents(3)
XTextWidth(3)
XTranslateCoordinates(3)
XUnmapEvent(3)
XUnmapWindow(3)
XVisibilityEvent(3)
XWarpPointer(3)
Xau(3)
XrmGetResource(3)
XrmInitialize(3)
XrmMergeDatabases(3)
XrmPutResource(3)
XrmUniqueQuark(3)
XtAddCallback(3)
XtAddEventHandler(3)
XtAddExposureToRegion(3)
XtAddGrab(3)
XtAppAddActions(3)
XtAppAddConverter(3)
XtSetArg(3)
XtSetKeyTranslator(3)
XtSetKeyboardFocus(3)
XtSetSensitive(3)
XtSetValues(3)
XtStringConversionWarning(3)
XtTranslateCoordinates(3)
appres(1)
bitmap(1)
execqt(1)
ico(1)
imake(1)
listres(1)
makedepend(1)
maze(1)
mkfontdir(1)
muncher(1)
mwm(1)
oclock(1)
plaid(1)
puzzle(1)
resize(1)
twm(1)
xauth(1)
xbiff(1)
xcalc(1)
xclipboard(1)
xclock(1)
xcutsel(1)
xdm(1)
xdpr(1)
xdpyinfo(1)
xedit(1)
xev(1)
xeyes(1)
xfd(1)
xfontsel(1)
xgc(1)
xhost(1)
xkill(1)
xload(1)
xlogo(1)
xlsatoms(1)
xlsclients(1)
xlsfonts(1)
xlswins(1)

XGravityEvent(3)
XIconifyWindow(3)
XIfEvent(3)
XInstallColormap(3)
XInternAtom(3)
XIntersectRegion(3)
XKeymapEvent(3)
XListFonts(3)
XLoadFont(3)
XLookupKeysym(3)
XMapEvent(3)
XMapRequestEvent(3)
XMapWindow(3)
XNextEvent(3)
XOpenDisplay(3)
XParseGeometry(3)
XPolygonRegion(3)
XPropertyEvent(3)

XtAppAddInput(3)
XtAppAddTimeOut(3)
XtAppAddWorkProc(3)
XtAppCreateShell(3)
XtAppError(3)
XtAppErrorMsg(3)
XtAppGetErrorDatabase(3)
XtAppGetSelectionTimeout(3)
XtAppNextEvent(3)
XtBuildEventMask(3)
XtCallAcceptFocus(3)
XtCallCallbacks(3)
XtClass(3)
XtConfigureWidget(3)
XtConvert(3)
XtCreateApplicationContext(3)
XtCreatePopupShell(3)
XtCreateWidget(3)

xmag(1)
xman(1)
xmodmap(1)
xpostage(1)
xpr(1)
xprop(1)
xrdb(1)
xrefresh(1)
xset(1)
xsetroot(1)
xstdcmap(1)
xterm(1)
xtraining(1)
xwd(1)
xwininfo(1)
xwud(1)

NAME

X - a portable, network-transparent window system

SYNOPSIS

The X Window System is a network transparent window system developed at MIT which runs on a wide range of computing and graphics machines. The core distribution from MIT has support for the following operating systems:

Ultrix 3.1 (Digital)
SunOS 4.0.3 (Sun)
HP-UX 6.5 (Hewlett-Packard)
Domain/OS 10.1 (HP/Apollo)
A/UX 1.1 (Apple)
AIX RT-2.2 and PS/2-1.1 (IBM)
AOS-4.3 (IBM)
UTEK 4.0 (Tektronix)
NEWS-OS 3.2 (Sony; client only)
UNICOS 5.0.1 (Cray; client only)
UNIX(tm) System V, Release 3.2 (AT&T 6386 WGS; client only)

It should be relatively easy to build the client-side software on a variety of other system. Commercial implementations are also available for a wide range of platforms.

The X Consortium requests that the following names be used when referring to this software:

X
X Window System
X Version 11
X Window System, Version 11
X11

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

DESCRIPTION

X Window System servers run on computers with bitmap displays. The server distributes user input to and accepts output requests from various client programs through a variety of different interprocess communication channels. Although the most common case is for the client programs to be running on the same machine as the server, clients can be run transparently from other machines (including machines with different architectures and operating systems) as well.

X supports overlapping hierarchical subwindows and text and graphics operations, on both monochrome and color displays. For a full explanation of the functions that are available, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, the *X Toolkit Intrinsics - C Language Interface* manual, and various toolkit documents.

The number of programs that use *X* is growing rapidly. Of particular interest are: a terminal emulator (*xterm*), a window manager (*twm*), a display manager (*xdm*), mail managing utilities (*xmh* and *xbiff*), a manual page browser (*xman*), a bitmap editor (*bitmap*), access control programs (*xauth* and *xhost*), user preference setting programs (*xrdb*, *xset*, *xsetroot*, and *xmodmap*), a load monitor (*xload*), clocks (*xclock* and *oclock*), a font displayer (*xfd*), utilities for listing information about fonts, windows, and displays (*xlsfonts*, *xfontsel*, *xlswins*, *xwininfo*, *xlsclients*, *xdpyinfo*, and *xprop*), a diagnostic for seeing what events are generated and when (*xev*), screen image manipulation utilities (*xwd*, *xwud*, *xpr*, and *xmag*), and various demos (*xeyes*, *ico*, *muncher*, *puzzle*, *xgc*, etc.).

Many other utilities, window managers, games, toolkits, etc. are available from the user-contributed software. See your site administrator for details.

STARTING UP

There are two main ways of getting the X server and an initial set of client applications started. The particular method used depends on what operating system you are running and on whether or not you use other window systems in addition to X.

xdm (the X Display Manager)

If you want to always have X running on your display, your site administrator can set your machine up to use the X Display Manager *xdm*. This program is typically started by the system at boot time and takes care of keeping the server running and getting users logged in. If you are running *xdm*, you will see a window on the screen welcoming you to the system and asking for your username and password. Simply type them in as you would at a normal terminal, pressing the Return key after each. If you make a mistake, *xdm* will display an error message and ask you to try again. After you have successfully logged in, *xdm* will start up your X environment. By default, if you have an executable file named *.xsession* in your home directory, *xdm* will treat it as a program (or shell script) to run to start up your initial clients (such as terminal emulators, clocks, a window manager, user settings for things like the background, the speed of the pointer, etc.). Your site administrator can provide details.

xinit (run manually from the shell)

Sites that support more than one window system might choose to use the *xinit* program for starting X manually. If this is true for your machine, your site administrator will probably have provided a program named "x11", "startx", or "xstart" that will do site-specific initialization (such as loading convenient default resources, running a window manager, displaying a clock, and starting several terminal emulators) in a nice way. If not, you can build such a script using the *xinit* program. This utility simply runs one user-specified program to start the server, runs another to start up any desired clients, and then waits for either to finish. Since either or both of the user-specified programs may be a shell script, this gives substantial flexibility at the expense of a nice interface. For this reason, *xinit* is not intended for end users.

DISPLAY NAMES

From the user's prospective, every X server has a *display name* of the form:

hostname:displaynumber.screennumber

This information is used by the application to determine how it should connect to the server and which screen it should use by default (on displays with multiple monitors):

hostname

The *hostname* specifies the name of the machine to which the display is physically connected. If the hostname is not given, the most efficient way of communicating to a server on the same machine will be used.

displaynumber

The phrase "display" is usually used to refer to collection of monitors that share a common keyboard and pointer (mouse, tablet, etc.). Most workstations tend to only have one keyboard, and therefore, only one display. Larger, multi-user systems, however, will frequently have several displays so that more than one person can be doing graphics work at once. To avoid confusion, each display on a machine is assigned a *display number* (beginning at 0) when the X server for that display is started. The display number must always be given in a display name.

screennumber

Some displays share a single keyboard and pointer among two or more monitors. Since each monitor has its own set of windows, each screen is assigned a *screen number* (beginning at 0) when the X server for that display is started. If the screen number is not

given, then screen 0 will be used.

On POSIX systems, the default display name is stored in your DISPLAY environment variable. This variable is set automatically by the *xterm* terminal emulator. However, when you log into another machine on a network, you'll need to set DISPLAY by hand to point to your display. For example,

```
% setenv DISPLAY myws:0
$ DISPLAY=myws:0; export DISPLAY
```

Finally, most X programs accept a command line option of **-display** *displayname* to temporarily override the contents of DISPLAY. This is most commonly used to pop windows on another person's screen or as part of a "remote shell" command to start an xterm pointing back to your display. For example,

```
% xeyes -display joesws:0 -geometry 1000x1000+0+0
% rsh big xterm -display myws:0 -ls </dev/null &
```

X servers listen for connections on a variety of different communications channels (network byte streams, shared memory, etc.). Since there can be more than one way of contacting a given server, The *hostname* part of the display name is used to determine the type of channel (also called a transport layer) to be used. The sample servers from MIT support the following types of connections:

local

The hostname part of the display name should be the empty string. For example: *:0*, *:1*, and *:0.1*. The most efficient local transport will be chosen.

TCP/IP

The hostname part of the display name should be the server machine's IP address name. Full Internet names, abbreviated names, and IP addresses are all allowed. For example: *expo.lcs.mit.edu:0*, *expo:0*, *18.30.0.212:0*, *bigmachine:1*, and *hydra:0.1*.

DECnet

The hostname part of the display name should be the server machine's nodename followed by two colons instead of one. For example: *myws::0*, *big::1*, and *hydra::0.1*.

ACCESS CONTROL

The sample server provides two types of access control: an authorization protocol which provides a list of "magic cookies" clients can send to request access, and a list of hosts from which connections are always accepted. *Xdm* initializes magic cookies in the server, and also places them in a file accessible to the user. Normally, the list of hosts from which connections are always accepted should be empty, so that only clients with are explicitly authorized can connect to the display. When you add entries to the host list (with *xhost*), the server no longer performs any authorization on connections from those machines. Be careful with this.

The file for authorization which both *xdm* and *Xlib* use can be specified with the environment variable **XAUTHORITY**, and defaults to the file **.Xauthority** in the home directory. *Xdm* uses **$HOME/.Xauthority** and will create it or merge in authorization records if it already exists when a user logs in.

To manage a collection of authorization files containing a collection of authorization records use *xauth*. This program allows you to extract records and insert them into other files. Using this, you can send authorization to remote machines when you login. As the files are machine-independent, you can also simply copy the files or use NFS to share them. If you use several machines, and share a common home directory with NFS, then you never really have to worry about authorization files, the system should work correctly by default. Note that magic cookies transmitted "in the clear" over NFS or using *ftp* or *rcp* can be "stolen" by a network eavesdropper, and as such may enable unauthorized access. In many environments this level of

security is not a concern, but if it is, you need to know the exact semantics of the particular magic cookie to know if this is actually a problem.

GEOMETRY SPECIFICATIONS

One of the advantages of using window systems instead of hardwired terminals is that applications don't have to be restricted to a particular size or location on the screen. Although the layout of windows on a display is controlled by the window manager that the user is running (described below), most X programs accept a command line argument of the form **-geometry** *WIDTHxHEIGHT+XOFF+YOFF* (where *WIDTH*, *HEIGHT*, *XOFF*, and *YOFF* are numbers) for specifying a preferred size and location for this application's main window.

The *WIDTH* and *HEIGHT* parts of the geometry specification are usually measured in either pixels or characters, depending on the application. The *XOFF* and *YOFF* parts are measured in pixels and are used to specify the distance of the window from the left or right and top and bottom edges of the screen, respectively. Both types of offsets are measured from the indicated edge of the screen to the corresponding edge of the window. The X offset may be specified in the following ways:

+XOFF The left edge of the window is to be placed *XOFF* pixels in from the left edge of the screen (i.e. the X coordinate of the window's origin will be *XOFF*). *XOFF* may be negative, in which case the window's left edge will be off the screen.

-XOFF The right edge of the window is to be placed *XOFF* pixels in from the right edge of the screen. *XOFF* may be negative, in which case the window's right edge will be off the screen.

The Y offset has similar meanings:

+YOFF The top edge of the window is to be *YOFF* pixels below the top edge of the screen (i.e. the Y coordinate of the window's origin will be *YOFF*). *YOFF* may be negative, in which case the window's top edge will be off the screen.

-YOFF The bottom edge of the window is to be *YOFF* pixels above the bottom edge of the screen. *YOFF* may be negative, in which case the window's bottom edge will be off the screen.

Offsets must be given as pairs; in other words, in order to specify either *XOFF* or *YOFF* both must be present. Windows can be placed in the four corners of the screen using the following specifications:

+0+0 upper left hand corner.

-0+0 upper right hand corner.

-0-0 lower right hand corner.

+0-0 lower left hand corner.

In the following examples, a terminal emulator will be placed in roughly the center of the screen and a load average monitor, mailbox, and clock will be placed in the upper right hand corner:

```
xterm -fn 6x10 -geometry 80x24+30+200 &
xclock -geometry 48x48-0+0 &
xload -geometry 48x48-96+0 &
xbiff -geometry 48x48-48+0 &
```

WINDOW MANAGERS

The layout of windows on the screen is controlled by special programs called *window managers*. Although many window managers will honor geometry specifications as given, others may choose to ignore them (requiring the user to explicitly draw the window's region on the screen with the pointer, for example).

Since window managers are regular (albeit complex) client programs, a variety of different user interfaces can be built. The core distribution comes with a window manager named *twm* which supports overlapping windows, popup menus, point-and-click or click-to-type input models, title bars, nice icons (and an icon manager for those who don't like separate icon windows).

Several other window managers are available in the user-contributed software: *gwm*, *m_swm*, *olwm*, and *tekwm*.

FONT NAMES

Collections of characters for displaying text and symbols in X are known as *fonts*. A font typically contains images that share a common appearance and look nice together (for example, a single size, boldness, slant, and character set). Similarly, collections of fonts that are based on a common type face (the variations are usually called roman, bold, italic, bold italic, oblique, and bold oblique) are called *families*.

Sets of font families of the same resolution (usually measured in dots per inch) are further grouped into *directories* (so named because they were initially stored in file system directories). Each directory contains a database which lists the name of the font and information on how to find the font. The server uses these databases to translate *font names* (which have nothing to do with file names) into font data.

The list of font directories in which the server looks when trying to find a font is controlled by the *font path*. Although most installations will choose to have the server start up with all of the commonly used font directories, the font path can be changed at any time with the *xset* program. However, it is important to remember that the directory names are on the **server**'s machine, not on the application's.

The default font path for the sample server contains three directories:

/usr/lib/X11/fonts/misc

This directory contains many miscellaneous fonts that are useful on all systems. It contains a small family of fixed-width fonts in pixel heights 5 through 10, a family of fixed-width fonts from Dale Schumacher in similar pixel heights, several Kana fonts from Sony Corporation, a Kanji font, the standard cursor font, two cursor fonts from Digital Equipment Corporation, and OPEN LOOK(tm) cursor and glyph fonts from Sun Microsystems. It also has font name aliases for the font names **fixed** and **variable**.

/usr/lib/X11/fonts/75dpi

This directory contains fonts contributed by Adobe Systems, Inc., Digital Equipment Corporation, Bitstream, Inc., Bigelow and Holmes, and Sun Microsystems, Inc. for 75 dots per inch displays. An integrated selection of sizes, styles, and weights are provided for each family.

/usr/lib/X11/fonts/100dpi

This directory contains 100 dots per inch versions of some of the fonts in the *75dpi* directory.

Font databases are created by running the *mkfontdir* program in the directory containing the source or compiled versions of the fonts (in both compressed and uncompressed formats). Whenever fonts are added to a directory, *mkfontdir* should be rerun so that the server can find the new fonts. To make the server reread the font database, reset the font path with the *xset* program. For example, to add a font to a private directory, the following commands could be used:

```
% cp newfont.snf ~/myfonts
% mkfontdir ~/myfonts
% xset fp rehash
```

The *xlsfonts* program can be used to list all of the fonts that are found in font databases in the current font path. Font names tend to be fairly long as they contain all of the information needed

to uniquely identify individual fonts. However, the sample server supports wildcarding of font names, so the full specification

-adobe-courier-medium-r-normal--10-100-75-75-m-60-iso8859-1

could be abbreviated as:

--courier-medium-r-normal--*-100-*-*-*-*-*-*-*

or, more tersely (but less accurately):

-courier-medium-r-normal---100-*

Because the shell also has special meanings for * and ?, wildcarded font names should be quoted:

% xlsfonts -fn '*-courier-medium-r-normal--*-100-*'

If more than one font in a given directory in the font path matches a wildcarded font name, the choice of which particular font to return is left to the server. However, if fonts from more than one directory match a name, the returned font will always be from the first such directory in the font path. The example given above will match fonts in both the *75dpi* and *100dpi* directories; if the *75dpi* directory is ahead of the *100dpi* directory in the font path, the smaller version of the font will be used.

COLOR NAMES

Most applications provide ways of tailoring (usually through resources or command line arguments) the colors of various elements in the text and graphics they display. Although black and white displays don't provide much of a choice, color displays frequently allow anywhere between 16 and 16 million different colors.

Colors are usually specified by their commonly-used names (for example, *red*, *white*, or *medium slate blue*). The server translates these names into appropriate screen colors using a color database that can usually be found in */usr/lib/X11/rgb.txt*. Color names are case-insensitive, meaning that *red*, *Red*, and *RED* all refer to the same color.

Many applications also accept color specifications of the following form:

#rgb
#rrggbb
#rrrgggbbb
#rrrrggggbbbb

where *r*, *g*, and *b* are hexadecimal numbers indicating how much *red*, *green*, and *blue* should be displayed (zero being none and ffff being on full). Each field in the specification must have the same number of digits (e.g., #rrgb or #gbb are not allowed). Fields that have fewer than four digits (e.g. #rgb) are padded out with zero's following each digit (e.g. #r000g000b000). The eight primary colors can be represented as:

black	#000000000000 (no color at all)
red	#ffff00000000
green	#0000ffff0000
blue	#00000000ffff
yellow	#ffffffff0000 (full red and green, no blue)
magenta	#ffff0000ffff
cyan	#0000ffffffff
white	#ffffffffffff (full red, green, and blue)

Unfortunately, RGB color specifications are highly unportable since different monitors produce different shades when given the same inputs. Similarly, color names aren't portable because there is no standard naming scheme and because the color database needs to be tuned for each monitor.

Application developers should take care to make their colors tailorable.

KEYS

The X keyboard model is broken into two layers: server-specific codes (called *keycodes*) which represent the physical keys, and server-independent symbols (called *keysyms*) which represent the letters or words that appear on the keys. Two tables are kept in the server for converting keycodes to keysyms:

modifier list

Some keys (such as Shift, Control, and Caps Lock) are known as *modifier* and are used to select different symbols that are attached to a single key (such as Shift-a generates a capital A, and Control-l generates a formfeed character ^L). The server keeps a list of keycodes corresponding to the various modifier keys. Whenever a key is pressed or released, the server generates an *event* that contains the keycode of the indicated key as well as a mask that specifies which of the modifier keys are currently pressed. Most servers set up this list to initially contain the various shift, control, and shift lock keys on the keyboard.

keymap table

Applications translate event keycodes and modifier masks into keysyms using a *keysym table* which contains one row for each keycode and one column for various modifier states. This table is initialized by the server to correspond to normal typewriter conventions, but is only used by client programs.

Although most programs deal with keysyms directly (such as those written with the X Toolkit Intrinsics), most programming libraries provide routines for converting keysyms into the appropriate type of string (such as ISO Latin-1).

OPTIONS

Most X programs attempt to use the same names for command line options and arguments. All applications written with the X Toolkit Intrinsics automatically accept the following options:

–display *display*

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

–geometry *geometry*

This option specifies the initial size and location of the window.

–bg *color*, **–background** *color*

Either option specifies the color to use for the window background.

–bd *color*, **–bordercolor** *color*

Either option specifies the color to use for the window border.

–bw *number*, **–borderwidth** *number*

Either option specifies the width in pixels of the window border.

–fg *color*, **–foreground** *color*

Either option specifies the color to use for text or graphics.

–fn *font*, **-font** *font*

Either option specifies the font to use for displaying text.

–iconic

This option indicates that the user would prefer that the application's windows initially not be visible as if the windows had be immediately iconified by the user. Window managers may choose not to honor the application's request.

–name

This option specifies the name under which resources for the application should be found. This option is useful in shell aliases to distinguish between invocations of an application, without resorting to creating links to alter the executable file name.

–rv, –reverse

Either option indicates that the program should simulate reverse video if possible, often by swapping the foreground and background colors. Not all programs honor this or implement it correctly. It is usually only used on monochrome displays.

+rv

This option indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

–selectionTimeout

This option specifies the timeout in milliseconds within which two communicating applications must respond to one another for a selection request.

–synchronous

This option indicates that requests to the X server should be sent synchronously, instead of asynchronously. Since *Xlib* normally buffers requests to the server, errors do not necessarily get reported immediately after they occur. This option turns off the buffering so that the application can be debugged. It should never be used with a working program.

–title *string*

This option specifies the title to be used for this window. This information is sometimes used by a window manager to provide some sort of header identifying the window.

–xnllanguage *language[_territory][.codeset]*

This option specifies the language, territory, and codeset for use in resolving resource and other filenames.

–xrm *resourcestring*

This option specifies a resource name and value to override any defaults. It is also very useful for setting resources that don't have explicit command line arguments.

RESOURCES

To make the tailoring of applications to personal preferences easier, X supports several mechanisms for storing default values for program resources (e.g. background color, window title, etc.) Resources are specified as strings of the form

*appname*subname*subsubname...: value*

that are read in from various places when an application is run. By convention, the application name is the same as the program name, but with the first letter capitalized (e.g. *Bitmap* or *Emacs*) although some programs that begin with the letter "x" also capitalize the second letter for historical reasons. The precise syntax for resources is:

ResourceLine	=	Comment | ResourceSpec
Comment	=	"!" string | <empty line>
ResourceSpec	=	WhiteSpace ResourceName WhiteSpace ":" WhiteSpace value
ResourceName	=	[Binding] ComponentName {Binding ComponentName}
Binding	=	"." | "*"
WhiteSpace	=	{" " | "\t"}
ComponentName	=	{"a"–"z" | "A"–"Z" | "0"–"9" | "_" | "-"}
value	=	string
string	=	{<any character not including "\n">}

Note that elements enclosed in curly braces ({...}) indicate zero or more occurrences of the enclosed elements

To allow values to contain arbitrary octets, the 4-character sequence *\nnn*, where n is a digit in the range of "0"–"7", is recognized and replaced with a single byte that contains this sequence interpreted as an octal number. For example, a value containing a NULL byte can be stored by specifying "\000".

The *Xlib* routine *XGetDefault(3X)* and the resource utilities within Xlib and the X Toolkit Intrinsics obtain resources from the following sources:

RESOURCE_MANAGER root window property
Any global resources that should be available to clients on all machines should be stored in the RESOURCE_MANAGER property on the root window using the *xrdb* program. This is frequently taken care of when the user starts up X through the display manager or *xinit*.

application-specific files
Programs that use the X Toolkit Intrinsics will also look in the directories named by the environment variable XUSERFILESEARCHPATH or the environment variable XAPPLRESDIR, plus directories in a standard place (usually under /usr/lib/X11/, but this can be overridden with the XFILESEARCHPATH environment variable) for application-specific resources. See the *X Toolkit Intrinsics - C Language Interface* manual for details.

XENVIRONMENT
Any user- and machine-specific resources may be specified by setting the XENVIRONMENT environment variable to the name of a resource file to be loaded by all applications. If this variable is not defined, a file named $*HOME*/.Xdefaults-*hostname* is looked for instead, where *hostname* is the name of the host where the application is executing.

–xrm *resourcestring*
Applications that use the X Toolkit Intrinsics can have resources specified from the command line. The *resourcestring* is a single resource name and value as shown above. Note that if the string contains characters interpreted by the shell (e.g., asterisk), they must be quoted. Any number of **–xrm** arguments may be given on the command line.

Program resources are organized into groups called *classes*, so that collections of individual resources (each of which are called *instances*) can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an upper case letter. Multiple word resources are concatenated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit Intrinsics will have at least the following resources:

background (class **Background**)
This resource specifies the color to use for the window background.

borderWidth (class **BorderWidth**)
This resource specifies the width in pixels of the window border.

borderColor (class **BorderColor**)
This resource specifies the color to use for the window border.

Most applications using the X Toolkit Intrinsics also have the resource **foreground** (class **Foreground**), specifying the color to use for text and graphics within the window.

By combining class and instance specifications, application preferences can be set quickly and easily. Users of color displays will frequently want to set Background and Foreground classes to particular defaults. Specific color instances such as text cursors can then be overridden without having to define all of the related resources. For example,

```
bitmap*Dashed: off
```

```
XTerm*cursorColor: gold
XTerm*multiScroll: on
XTerm*jumpScroll: on
XTerm*reverseWrap: on
XTerm*curses: on
XTerm*Font: 6x10
XTerm*scrollBar: on
XTerm*scrollbar*thickness: 5
XTerm*multiClickTime: 500
XTerm*charClass: 33:48,37:48,45-47:48,64:48
XTerm*cutNewline: off
XTerm*cutToBeginningOfLine: off
XTerm*titeInhibit: on
XTerm*ttyModes: intr ^c erase ^? kill ^u
XLoad*Background: gold
XLoad*Foreground: red
XLoad*highlight: black
XLoad*borderWidth: 0
emacs*Geometry: 80x65-0-0
emacs*Background: #5b7686
emacs*Foreground: white
emacs*Cursor: white
emacs*BorderColor: white
emacs*Font: 6x10
xmag*geometry: -0-0
xmag*borderColor: white
```

If these resources were stored in a file called *.Xresources* in your home directory, they could be added to any existing resources in the server with the following command:

```
% xrdb -merge $HOME/.Xresources
```

This is frequently how user-friendly startup scripts merge user-specific defaults into any site-wide defaults. All sites are encouraged to set up convenient ways of automatically loading resources. See the *Xlib* manual section *Using the Resource Manager* for more information.

EXAMPLES

The following is a collection of sample command lines for some of the more frequently used commands. For more information on a particular command, please refer to that command's manual page.

```
% xrdb -load $HOME/.Xresources
% xmodmap -e "keysym BackSpace = Delete"
% mkfontdir /usr/local/lib/X11/otherfonts
% xset fp+ /usr/local/lib/X11/otherfonts
% xmodmap $HOME/.keymap.km
% xsetroot -solid '#888'
% xset b 100 400 c 50 s 1800 r on
% xset q
% twm
% xmag
% xclock -geometry 48x48-0+0 -bg blue -fg white
% xeyes -geometry 48x48-48+0
% xbiff -update 20
% xlsfonts '*helvetica*'
```

```
% xlswins -l
% xwininfo -root
% xdpyinfo -display joesworkstation:0
% xhost -joesworkstation
% xrefresh
% xwd | xwud
% bitmap companylogo.bm 32x32
% xcalc -bg blue -fg magenta
% xterm -geometry 80x66-0-0 -name myxterm $*
```

DIAGNOSTICS

A wide variety of error messages are generated from various programs. Various toolkits are encouraged to provide a common mechanism for locating error text so that applications can be tailored easily. Programs written to interface directly to the *Xlib* C language library are expected to do their own error checking.

The default error handler in *Xlib* (also used by many toolkits) uses standard resources to construct diagnostic messages when errors occur. The defaults for these messages are usually stored in */usr/lib/X11/XErrorDB*. If this file is not present, error messages will be rather terse and cryptic.

When the X Toolkit Intrinsics encounter errors converting resource strings to the appropriate internal format, no error messages are usually printed. This is convenient when it is desirable to have one set of resources across a variety of displays (e.g. color vs. monochrome, lots of fonts vs. very few, etc.), although it can pose problems for trying to determine why an application might be failing. This behavior can be overridden by the setting the *StringConversionsWarning* resource.

To force the X Toolkit Intrinsics to always print string conversion error messages, the following resource should be placed at the top of the file that gets loaded onto the RESOURCE_MANAGER property using the *xrdb* program (frequently called *.Xresources* or *.Xres* in the user's home directory):

```
*StringConversionWarnings: on
```

To have conversion messages printed for just a particular application, the appropriate instance name can be placed before the asterisk:

```
xterm*StringConversionWarnings: on
```

BUGS

If you encounter a **repeatable** bug, please contact your site administrator for instructions on how to submit an X Bug Report.

SEE ALSO

XConsortium(1), XStandards(1), appres(1), bdftosnf(1), bitmap(1), imake(1), listres(1), maze(1), mkfontdir(1), muncher(1), oclock(1), puzzle(1), resize(1), showsnf(1), twm(1), xauth(1), xbiff(1), xcalc(1), xclipboard(1), xclock(1), xditview(1), xdm(1), xdpyinfo(1), xedit(1), xev(1), xeyes(1), xfd(1), xfontsel(1), xhost(1), xinit(1), xkill(1), xload(1), xlogo(1), xlsatoms(1), xlsclients(1), xlsfonts(1), xlswins(1), xmag(1), xman(1), xmh(1), xmodmap(1), xpr(1), xprop(1), xrdb(1), xrefresh(1), xset(1), xsetroot(1), xstdcmap(1), xterm(1), xwd(1), xwininfo(1), xwud(1), Xserver(1), Xapollo(1), Xcfbpmax(1), Xhp(1), Xibm(1), XmacII(1), Xmfbpmax(1), Xqdss(1), Xqvss(1), Xsun(1), Xtek(1), kbd_mode(1), todm(1), tox(1), biff(1), init(8), ttys(5), *Xlib - C Language X Interface*, *X Toolkit Intrinsics - C Language Interface*, and *Using and Specifying X Resources*

COPYRIGHT

TRADEMARKS

UNIX and OPEN LOOK are trademarks of AT&T. X Window System is a trademark of MIT.

AUTHORS

A cast of thousands, literally. The X distribution is brought to you by the MIT X Consortium. The staff members at MIT responsible for this release are: Donna Converse (MIT X Consortium), Jim Fulton (MIT X Consortium), Michelle Leger (MIT X Consortium), Keith Packard (MIT X Consortium), Chris Peterson (MIT X Consortium), Bob Scheifler (MIT X Consortium), and Ralph Swick (Digital/MIT Project Athena).

NAME

XConsortium - MIT X Consortium

SYNOPSIS

X11R4 is brought to you by the MIT X Consortium.

DESCRIPTION

X Window System research began in the summer of 1984 at MIT as an informal joint undertaking between the Laboratory for Computer Science and Project Athena. Since that time, researchers and engineers at numerous other universities and companies have been involved in the design and implementation.

The MIT X Consortium was formed in January of 1988 to further the development of the X Window System. It is part of the Laboratory for Computer Science at MIT, and has as its major goal the promotion of cooperation within the computer industry in the creation of standard software interfaces at all layers in the X Window System environment. MIT's role is to provide the vendor-neutral architectural and administrative leadership required to make this work. The Consortium is financially self-supporting, with membership open to any organization. There are two categories of membership, Member (for large organizations) and Affiliate (for smaller organizations). A list of members as of the public release of X11R4 is given below.

The Director of the X Consortium acts as the ultimate architect for all X specifications and software. The activities of the Consortium are overseen by an MIT Steering Committee, which helps set policy and provides strategic guidance and review of the Consortium's activities. An Advisory Committee made up of member representatives meets regularly to review the Consortium's plans, assess its progress, and suggest future directions.

Most of the Consortium's activities take place via electronic mail, with meetings when required. As designs and specifications take shape, interest groups are formed from experts in the participating organizations. Typically a small multi-organization architecture team leads the design, with others acting as close observers and reviewers. Once a complete specification is produced, it may be submitted for formal technical review by the Consortium as a proposed standard. The standards process includes public review (outside the Consortium) and a demonstration of proof of concept, typically established with a public, portable implementation of the specification, (although other methods are possible on a case by case basis).

The MIT staff of the Consortium also maintains a software and documentation collection, containing both Consortium-developed and user-contributed materials, and endeavors to make periodic distributions of this collection available to the public without license and for a minimal fee. X11R4 is the fourth such distribution. The Consortium also sponsors an annual conference, open to the public, to promote the exchange of technical information about X.

STAFF

The Director of the X Consortium is Bob Scheifler, rws@expo.lcs.mit.edu. The MIT staff members are as follows.

Donna Converse, converse@expo.lcs.mit.edu
Jim Fulton, jim@expo.lcs.mit.edu
Michelle Leger, michelle@expo.lcs.mit.edu
Keith Packard, keith@expo.lcs.mit.edu
Chris Peterson, kit@expo.lcs.mit.edu

Ralph Swick, swick@athena.mit.edu, an employee of Digital Equipment Corporation working at MIT Project Athena, also works directly with the Consortium staff.

POSTAL ADDRESS

The MIT X Consortium can be reached by writing to

Bob Scheifler
MIT X Consortium
Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

MEMBERS

Apple Computer, Inc.
American Telephone & Telegraph Company
Bull S.A.
Control Data Corporation
Cray Research, Inc.
Data General Corporation
Digital Equipment Corporation
Eastman Kodak Company
Fujitsu Limited
Hewlett-Packard Company
International Business Machines Corporation
Mitsubishi Electric Corporation
Motorola, Inc.
NEC Corporation
NCR Corporation
Nippon Telegraph and Telephone Corporation
OMRON Corporation
Prime Computer, Inc.
Rich, Inc.
Sequent Computer Systems Inc.
Siemens AG
Silicon Graphics, Inc.
Sony Corporation
Sun Microsystems, Inc.
Tektronix, Inc.
Texas Instruments, Inc.
Unisys Corporation
Wang Laboratories, Inc.
Xerox Corporation

AFFILIATES

ACER America Corporation
Adobe Systems
Advanced Graphics Engineering
Carnegie Mellon University
CETIA - Compagnie Europeene des Techniques de l'Ingenierie Assistee
Codonics, Inc.
Evans & Sutherland
GfxBase
INESC - Instituto de Engenharia de Sistemas e Computadores
Integrated Solutions
Interactive Systems Corporation
Interactive Development Environments
Integrated Computer Solutions, Inc.

Key Systems Engineering Corp.
Jupiter Systems
University of Kent at Canterbury
Landmark Graphics Corporation
Locus Computing
University of Lowell
Matrox International
Megatek Corporation
MIPS Computer Systems
MITRE Corporation
Network Computing Devices
Nova Graphics International
Open Software Foundation
O'Reilly & Associates, Inc.
PCS Computer Systeme GmbH
Ramtek Corporation
Samsung Software America
SGIP - Societe de Gestion et d'Informatique Publicis
Software Productivity Consortium
Solbourne Computer Inc.
Spectragraphics Corporation
Stanford University
Stardent Computer
Tatung Science and Technology
UNICAD, Inc.
Visual Technology Inc.
X/Open Company Ltd.

NAME

XStandards - X Standards

SYNOPSIS

The major goal of the MIT X Consortium is to promote cooperation within the computer industry in the creation of standard software interfaces at all layers in the X Window System environment. The status of various standards and proposed standards, and of the software in the X11R4 distribution, is explained below.

STANDARDS

The following documents are MIT X Consortium standards:

X Window System Protocol
X Version 11, Release 4
Robert W. Scheifler

Xlib - C Language X Interface
X Version 11, Release 4
James Gettys, Robert W. Scheifler, Ron Newman

X Toolkit Intrinsics - C Language Interface
X Version 11, Release 4
Joel McCormack, Paul Asente, Ralph R. Swick

Bitmap Distribution Format
Version 2.1

Inter-Client Communication Conventions Manual
Version 1.0
David S. H. Rosenthal

Compound Text Encoding
Version 1.1
Robert W. Scheifler

X Logical Font Description Conventions
Version 1.3
Jim Flowers

X Display Manager Control Protocol
Version 1.0
Keith Packard

X11 Nonrectangular Window Shape Extension
Version 1.0
Keith Packard

DRAFT STANDARDS

The following documents are draft standards of the MIT X Consortium. To become standards, further "proof of concept" is required, in the form of working implementations. The specifications may be subject to incompatible changes if implementation efforts uncover significant problems.

PEX Protocol Specification
Version 4.0P

Randi J. Rost (editor)

Extending X for Double-Buffering, Multi-Buffering, and Stereo
Version 3.2
Jeffrey Friedberg, Larry Seiler, Jeff Vroom

PUBLIC REVIEW DRAFTS

The following documents are out for Public Review for adoption as MIT X Consortium standards.

X11 Input Extension Protocol Specification
Public Review Draft
George Sachs, Mark Patrick

X11 Input Extension Library Specification
Public Review Draft
Mark Patrick, George Sachs

INCLUDE FILES

The following include files are part of the Xlib standard. The C++ support in these header files is experimental, and is not yet part of the standard.

<X11/X.h>
<X11/Xatom.h>
<X11/Xproto.h>
<X11/Xprotostr.h>
<X11/keysym.h>
<X11/keysymdef.h>
<X11/Xlib.h>
<X11/Xresource.h>
<X11/Xutil.h>
<X11/cursorfont.h>
<X11/X10.h>
<X11/Xlibint.h>

The following include files are part of the X Toolkit Intrinsics standard. The C++ support in these header files is experimental, and is not yet part of the standard.

<X11/Composite.h>
<X11/CompositeP.h>
<X11/ConstrainP.h>
<X11/Constraint.h>
<X11/Core.h>
<X11/CoreP.h>
<X11/Intrinsic.h>
<X11/IntrinsicP.h>
<X11/Object.h>
<X11/ObjectP.h>
<X11/Quarks.h>
<X11/RectObj.h>
<X11/RectObjP.h>
<X11/Shell.h>
<X11/ShellP.h>

<X11/StringDefs.h>
<X11/Vendor.h>
<X11/VendorP.h>

The following include file is part of the Nonrectangular Window Shape Extension standard.

<X11/extensions/shape.h>

The following include file is part of the Multi-Buffering draft standard.

<X11/extensions/multibuf.h>

NON STANDARDS

The X11R4 distribution contains *sample* implementations, not *reference* implementations. Although much of the code is believed to be correct, the code should be assumed to be in error wherever it conflicts with the specification.

At the public release of X11R4, the only MIT X Consortium standards are the ones listed above. No other documents, include files, or software in X11R4 carry special status within the X Consortium. For example, none of the following are standards: internal interfaces of the sample server; the MIT-SHM extension; the Input Synthesis extension; the Athena Widget Set; the Xmu library; the Xau library; CLX, the Common Lisp interface to X (although a Consortium review is finally expected to begin); the RGB database; the fonts distributed with X11R4; the applications distributed with X11R4; the include files <X11/XWDFile.h> and <X11/Xos.h>; the bitmap files in <X11/bitmaps>.

NAME

appres - list application resource database

SYNOPSIS

appres [[classname [instancename]] [-xrm resource ...]

DESCRIPTION

The *appres* program prints the resources seen by an application of the specified *class* and *instance* name. It is used to determine which resources a particular program would load. For example,

```
% appres XTerm
```

would list the resources that any *xterm* program would load. To also match particular instance names,

```
% appres myxterm XTerm
```

If no application class is specified, the class *-NoSuchClass-* (which should have no defaults) is used.

SEE ALSO

X(1), xrdb(1), listres(1)

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

appres is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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 */tmp/filename* 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 Pixmaps (often used for window backgrounds and borders). Applications writers should be careful to understand the difference between Bitmaps and Pixmaps 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*), *XmuReadBitmapDataFromFile*

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

AUTHOR

bitmap by Ron Newman, MIT Project Athena; documentation, *bmtoa,* and *atobm* by Jim Fulton, MIT X Consortium.

NOTE

bitmap is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

execqt - execute a command in the background

SYNOPSIS

execqt *command* [*arguments* ...]

DESCRIPTION

execqt is a simple program that executes *command* "in the background" in the style of system daemon programs. The given *command* is disassociated from the controlling terminal and executed. The standard output and standard error of *command* is lost. Any command arguments will be passed to *command*.

execqt is especially useful for starting X applications on remote machines via *rsh*(1).

ENVIRONMENT

execqt will use the environment variable "PATH" to search for the specified *command*. When executing X programs, *execqt* will use the environment variable "DISPLAY" to specify which bitmap display to use.

DIAGNOSTICS

If *command* cannot be executed, an appropriate error message is logged by *syslog*(8).

EXAMPLE

To execute *xterm*(1) on remote host "convexa":

```
rsh convexa execqt xterm -ls -display works:0
```

SEE ALSO

X(1), rsh(1), xterm(1), syslog(3), syslogd(8)

COPYRIGHT

AUTHORS

Tom Christiansen, CONVEX Computer Corporation.

NOTES

execqt is an optional product; contact your CONVEX sales representative for more information.

NAME

ico – animate an icosahedron or other polyhedron

SYNOPSIS

ico [-display display] [-geometry geometry] [-r] [-d pattern] [-i] [-dbl] [-faces] [-noedges] [-sleep n] [-obj object] [-objhelp] [-colors color-list]

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

-r Display on the root window instead of creating a new window.

-d pattern
Specify a bit pattern for drawing dashed lines for wire frames.

-i Use inverted colors for wire frames.

-dbl Use double buffering on the display. This works for either wire frame or solid fill drawings. For solid fill drawings, using this switch results in substantially smoother movement. Note that this requires twice as many bit planes as without double buffering. Since some colors are typically allocated by other programs, most eight-bit-plane displays will probably be limited to eight colors when using double buffering.

-faces Draw filled faces instead of wire frames.

-noedges
Don't draw the wire frames. Typically used only when -faces is used.

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

-obj *object*
Specify what object to draw. If no object is specified, an icosahedron is drawn.

-objhelp
Print out a list of the available objects, along with information about each object.

-colors *color color ...*
Specify what colors should be used to draw the filled faces of the object. If less colors than faces are given, the colors are reused.

ADDING POLYHEDRA

If you have the source to ico, it is very easy to add more polyhedra. Each polyhedron is defined in an include file by the name of objXXX.h, where XXX is something related to the name of the polyhedron. The format of the include file is defined in the file polyinfo.h. Look at the file objcube.h to see what the exact format of an objXXX.h file should be, then create your objXXX.h file in that format.

After making the new objXXX.h file (or copying in a new one from elsewhere), simply do a 'make depend'. This will recreate the file allobjs.h, which lists all of the objXXX.h files. Doing a 'make' after this will rebuild ico with the new object information.

SEE ALSO

X(1)

BUGS

A separate color cell is allocated for each name in the -colors list, even when the same name may be specified twice.

NOTE

ico is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

imake – C preprocessor interface to the make utility

SYNOPSIS

imake [-D*define*] [-I*dir*] [-T*template*] [-f *filename*] [-s *filename*] [-e] [-v]

DESCRIPTION

Imake is used to generate *Makefiles* from a template, a set of *cpp* macro functions, and a per-directory input file called an *Imakefile*. This allows machine dependencies (such has compiler options, alternate command names, and special *make* rules) to be kept separate from the descriptions of the various items to be built.

OPTIONS

The following command line options may be passed to *imake*:

–D*define*

This option is passed directly to *cpp*. It is typically used to set directory-specific variables. For example, the X Window System uses this flag to set *TOPDIR* to the name of the directory containing the top of the core distribution and *CURDIR* to the name of the current directory, relative to the top.

–I*directory*

This option is passed directly to *cpp*. It is typically used to indicate the directory in which the *imake* template and configuration files may be found.

–T*template*

This option specifies the name of the master template file (which is usually located in the directory specified with *-I*) used by *cpp*. The default is *Imake.tmpl*.

–f *filename*

This option specifies the name of the per-directory input file. The default is *Imakefile*.

–s *filename*

This option specifies the name of the *make* description file to be generated but *make* should not be invoked. If the *filename* is a dash (-), the output is written to *stdout*. The default is to generate, but not execute, a *Makefile*.

–e This option indicates the *imake* should execute the generated *Makefile*. The default is to leave this to the user.

–v This option indicates that *imake* should print the *cpp* command line that it is using to generate the *Makefile*.

HOW IT WORKS

Imake invokes *cpp* with any *-I* or *-D* flags passed on the command line and passes it the following 3 lines:

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

where *Imake.tmpl* and *Imakefile* may be overridden by the *-T* and *-f* command options, respectively. If the *Imakefile* contains any lines beginning with a '#' character that is not followed by a *cpp* directive (**#include**, **#define**, **#undef**, **#ifdef**, **#else**, **#endif**, or **#if**), *imake* will make a temporary *makefile* in which the '#' lines are prepended with the string "/**/" (so that *cpp* will copy the line into the *Makefile* as a comment).

The *Imakefile* reads in file containing machine-dependent parameters (specified as *cpp* symbols), a site-specific parameters file, a file containing *cpp* macro functions for generating *make* rules, and finally the *Imakefile* (specified by INCLUDE_IMAKEFILE) in the current directory. The *Imakefile* uses the macro functions to indicate what targets should be built; *imake* takes care of generating

the appropriate rules.

The rules file (usually named *Imake.rules* in the configuration directory) contains a variety of *cpp* macro functions that are configured according to the current platform. *Imake* replaces any occurrences of the string "@@" with a newline to allow macros that generate more than one line of *make* rules. For example, the macro

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

when called with *program_target(foo, foo1.o foo2.o)* will expand to

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

On systems whose *cpp* reduces multiple tabs and spaces to a single space, *imake* attempts to put back any necessary tabs (*make* is very picky about the difference between tabs and spaces). For this reason, colons (:) in command lines must be preceded by a backslash (\).

USE WITH THE X WINDOW SYSTEM

The X Window System uses *imake* extensively, for both full builds within the source tree and external software. As mentioned above, two special variables, *TOPDIR* and *CURDIR* set to make referencing files using relative path names easier. For example, the following command is generated automatically to build the *Makefile* in the directory *lib/X/* (relative to the top of the sources):

```
% ../../../config/imake -I../../../config \
                -DTOPDIR=../../. -DCURDIR=./lib/X
```

When building X programs outside the source tree, a special symbol *UseInstalled* is defined and *TOPDIR* and *CURDIR* are omitted. If the configuration files have been properly installed, the script *xmkmf(1)* may be used to specify the proper options:

```
% xmkmf
```

The command *make Makefiles* can then be used to generate *Makefiles* in any subdirectories.

FILES

/usr/tmp/tmp-imake.*nnnnnn*	temporary input file for cpp
/usr/tmp/tmp-make.*nnnnnn*	temporary input file for make
/lib/cpp	default C preprocessor

SEE ALSO

make(1)

S. I. Feldman *Make - A Program for Maintaining Computer Programs*

ENVIRONMENT VARIABLES

The following environment variables may be set, however their use is not recommended as they introduce dependencies that are not readily apparent when *imake* is run:

IMAKEINCLUDE

If defined, this should be a valid include argument for the C preprocessor. E.g. "-I/usr/include/local". Actually, any valid *cpp* argument will work here.

IMAKECPP

If defined, this should be a valid path to a preprocessor program. E.g. "/usr/local/cpp".

By default, *imake* will use /lib/cpp.

IMAKEMAKE

If defined, this should be a valid path to a make program. E.g. "/usr/local/make". By default, *imake* will use whatever *make* program is found using *execvp(3)*.

BUGS

Comments should be preceded by "/**/#" to protect them from *cpp*.

AUTHOR

Todd Brunhoff, Tektronix and MIT Project Athena; Jim Fulton, MIT X Consortium

NAME

listres - list resources in widgets

SYNOPSIS

listres [-option ...]

DESCRIPTION

The *listres* program generates a list of a widget's resource database. The class in which each resource is first defined, the instance and class name, and the type of each resource is listed. If no specific widgets or the *-all* switch are given, a two-column list of widget names and their class hierarchies is printed.

OPTIONS

Listres accepts all of the standard toolkit command line options along with those listed below:

-all This option indicates that *listres* should print information for all known widgets and objects.

-nosuper

This option indicates that resources that are inherited from a superclass should not be listed. This is useful for determining which resources are new to a subclass.

-variable

This option indicates that widgets should be identified by the names of the class record variables rather than the class name given in the variable. This is useful for distinguishing subclasses that have the same class name as their superclasses.

-top *name*

This option specifies the name of the widget to be treated as the top of the hierarchy. Case is not significant, and the name may match either the class variable name or the class name. The default is "core".

-format *printf-string*

This option specifies the *printf*-style format string to be used to print out the name, instance, class, and type of each resource.

X DEFAULTS

To be written.

SEE ALSO

X(1), xrdb(1), appropriate widget documents

BUGS

On operating systems that do not support dynamic linking of run-time routines, this program must have all of its known widgets compiled in. The sources provide several tools for automating this process for various widget sets.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

listres is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

makedepend – create dependencies in makefiles

SYNOPSIS

makedepend [**–Dname=def**] [**–Dname**] [**–Iincludedir**] [**–fmakefile**] [**–oobjsuffix**] [**–sstring**] [**–wwidth**] [– – **otheroptions** – –] sourcefile ...

DESCRIPTION

Makedepend reads each *sourcefile* in sequence and parses it like a C-preprocessor, processing all *#include, #define, #undef, #ifdef, #ifndef, #endif, #if* and *#else* directives so that it can correctly tell which *#include,* directives would be used in a compilation. Any *#include,* directives can reference files having other *#include* directives, and parsing will occur in these files as well.

Every file that a *sourcefile* includes, directly or indirectly, is what **makedepend** calls a "dependency". These dependencies are then written to a *makefile* in such a way that **make(1)** will know which object files must be recompiled when a dependency has changed.

By default, **makedepend** places its output in the file named *makefile* if it exists, otherwise *Makefile.* An alternate makefile may be specified with the *-f* option. It first searches the makefile for the line

```
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

or one provided with the *-s* option, as a delimiter for the dependency output. If it finds it, it will delete everything following this to the end of the makefile and put the output after this line. If it doesn't find it, the program will append the string to the end of the makefile and place the output following that. For each *sourcefile* appearing on the command line, **makedepend** puts lines in the makefile of the form

```
sourcefile.o: dfile ...
```

Where "sourcefile.o" is the name from the command line with its suffix replaced with ".o", and "dfile" is a dependency discovered in a *#include* directive while parsing *sourcefile* or one of the files it included.

EXAMPLE

Normally, **makedepend** will be used in a makefile target so that typing "make depend" will bring the dependencies up to date for the makefile. For example,

```
SRCS = file1.c file2.c ...
CFLAGS = -O -DHACK -I../foobar -xyz
depend:
        makedepend -- $(CFLAGS) -- $(SRCS)
```

OPTIONS

Makedepend will ignore any option that it does not understand so that you may use the same arguments that you would for **cc(1).**

–Dname=def or –Dname

Define. This places a definition for *name* in **makedepend's** symbol table. Without *=def* the symbol becomes defined as "1".

–Iincludedir

Include directory. This option tells **makedepend** to prepend *includedir* to its list of directories to search when it encounters a *#include* directive. By default, **makedepend** only searches /usr/include.

–fmakefile

Filename. This allows you to specify an alternate makefile in which **makedepend** can place its output.

–oobjsuffix
Object file suffix. Some systems may have object files whose suffix is something other than ".o". This option allows you to specify another suffix, such as ".b" with *-o.b* or ":obj" with *-o:obj* and so forth.

–sstring
Starting string delimiter. This option permits you to specify a different string for **makedepend** to look for in the makefile.

–wwidth
Line width. Normally, **makedepend** will ensure that every output line that it writes will be no wider than 78 characters for the sake of readability. This option enables you to change this width.

– – options – –
If **makedepend** encounters a double hyphen (– –) in the argument list, then any unrecognized argument following it will be silently ignored; a second double hyphen terminates this special treatment. In this way, **makedepend** can be made to safely ignore esoteric compiler arguments that might normally be found in a CFLAGS **make** macro (see the **EXAMPLE** section above). All options that **makedepend** recognizes and appear between the pair of double hyphens are processed normally.

ALGORITHM

The approach used in this program enables it to run an order of magnitude faster than any other "dependency generator" I have ever seen. Central to this performance are two assumptions: that all files compiled by a single makefile will be compiled with roughly the same *-I* and *-D* options; and that most files in a single directory will include largely the same files.

Given these assumptions, **makedepend** expects to be called once for each makefile, with all source files that are maintained by the makefile appearing on the command line. It parses each source and include file exactly once, maintaining an internal symbol table for each. Thus, the first file on the command line will take an amount of time proportional to the amount of time that a normal C preprocessor takes. But on subsequent files, if it encounter's an include file that it has already parsed, it does not parse it again.

For example, imagine you are compiling two files, *file1.c* and *file2.c*, they each include the header file *header.h*, and the file *header.h* in turn includes the files *def1.h* and *def2.h*. When you run the command

```
makedepend file1.c file2.c
```

makedepend will parse *file1.c* and consequently, *header.h* and then *def1.h* and *def2.h*. It then decides that the dependencies for this file are

```
file1.o: header.h def1.h def2.h
```

But when the program parses *file2.c* and discovers that it, too, includes *header.h*, it does not parse the file, but simply adds *header.h*, *def1.h* and *def2.h* to the list of dependencies for *file2.o*.

SEE ALSO

cc(1), make(1)

BUGS

If you do not have the source for cpp, the Berkeley C preprocessor, then **makedepend** will be compiled in such a way that all *#if* directives will evaluate to "true" regardless of their actual value. This may cause the wrong *#include* directives to be evaluated. **Makedepend** should simply have its own parser written for *#if* expressions.

Imagine you are parsing two files, say *file1.c* and *file2.c*, each includes the file *def.h*. The list of files that *def.h* includes might truly be different when *def.h* is included by *file1.c* than when it is included by *file2.c*. But once **makedepend** arrives at a list of dependencies for a file, it is cast in concrete.

AUTHOR

Todd Brunhoff, Tektronix, Inc. and MIT Project Athena

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...

LIMITATIONS

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

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

NOTE

maze is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

mkfontdir - create fonts.dir file from directory of font files.

SYNOPSIS

mkfontdir [directory-names]

DESCRIPTION

Mkfontdir For each directory argument, mkfontdir reads all of the font files in the directory searching for properties named "FONT", or (failing that) the name of the file stripped of its suffix. These are used as font names, which are written out to the file "fonts.dir" in the directory along with the name of the font file.

The kinds of font files read by mkfontdir depends on configuration parameters, but typically include SNF (suffix ".snf"), compressed SNF (suffix ".snf.Z"), BDF (suffix ".bdf"), and compressed BDF (suffix ".bdf.Z"). If a font exists in multiple formats, the most efficient format will be used.

FONT NAME ALIASES

The file "fonts.alias" which can be put in any directory of the font-path is used to map new names to existing fonts, and should be edited by hand. The format is straight forward enough, two white-space separated columns, the first containing aliases and the second containing font-name patterns.

When a font alias is used, the name it references is search for in the normal manner, looking through each font directory in turn. This means that the aliases need not mention fonts in the same directory as the alias file.

To embed white-space in either name, simply enclose them in double-quote marks, to embed double-quote marks (or any other character), precede them with back-slash:

"magic-alias with spaces"	"\"font\name\" with quotes"
regular-alias	fixed

If the string "FILE_NAMES_ALIASES" stands alone on a line, each file-name in the directory (stripped of it's .snf suffix) will be used as an alias for that font.

USAGE

Xserver(1) looks for both "fonts.dir" and "fonts.alias" in each directory in the font path each time it is set (see xset(1)).

SEE ALSO

X(1), Xserver(1), xset(1)

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

NOTE

muncher is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

mwm - A Window Manager

SYNOPSIS

mwm [*options*]

DESCRIPTION

mwm is an X11 client that provides window management functionality and some session management functionality. It provides functions that facilitate control (by the user and the programmer) of elements of window states such as placement, size, icon/normal display, input focus ownership, etc. It also provides session management functions such as stopping a client.

OPTIONS

–display *display*

This option specifies the display to use; see *X(1)*.

–xrm *resourcestring*

This option specifies a resource string to use.

Appearance

The following sections describe the basic default behaviors of windows, icons, the icon box, input focus, and window stacking. The appearance and behavior of the window manager can be altered by changing the configuration of specific resources. Resources are defined under the heading "X DEFAULTS."

Windows

Default **mwm** window frames have distinct components with associated functions:

Title Area	In addition to displaying the client's title, the title area is used to move the window. To move the window, place the pointer over the title area, press button 1 and drag the window to a new location. A wire frame is moved during the drag to indicate the new location. When the button is released, the window is moved to the new location.
Title Bar	The title bar includes the title area, the minimize button, the maximize button and the window menu button.
Minimize Button	To turn the window back into its icon, do a button 1 click on the minimize button (the frame box with a *small* square in it).
Maximize Button	To make the window fill the screen (or enlarge to the largest size allowed by the configuration files), do a button 1 click on the maximize button (the frame box with a *large* square in it).
Window Menu Button	The window menu button is the frame box with a horizontal bar in it. To pop up the window menu, press button 1. While pressing, drag the pointer on the menu to your selection, then release the button when your selection is highlighted. Alternately, you can click button 1 to pop up the menu and keep it posted; then position the pointer and select.

Default Window Menu		
Selection	**Accelerator**	**Description**
Restore	Alt+F5	Inactive (not an option for windows).
Move	Alt+F7	Allows the window to be moved with keys or mouse.
Size	Alt+F8	Allows the window to be resized.
Minimize	Alt+F9	Turns the window into an icon.
Maximize	Alt+F10	Makes the window fill the screen.
Lower	Alt+F11	Moves window to bottom of window stack.
Close	Alt+F4	Removes client from **mwm** management.

Resize Border Handles

To change the size of a window, move the pointer over a resize border handle (the cursor will change), press button 1, and drag the window to a new size. When the button is released, the window is resized. While dragging is being done, a rubber-band outline is displayed to indicate the new window size.

Matte

An optional matte decoration can be added between the client area and the window frame. A matte is not actually part of the window frame. There is no functionality associated with a matte.

Icons

Icons are small graphic representations of windows. A window can be minimized (iconified) using the minimize button on the window frame. Icons provide a way to reduce clutter on the screen.

Pressing mouse button 1 when the pointer is over an icon will cause the icon's window menu to pop up. Releasing the button (press + release without moving mouse = click) will cause the menu to stay posted. The menu contains the following selections:

Icon Window Menu		
Selection	**Accelerator**	**Description**
Restore	Alt+F5	Opens the associated window.
Move	Alt+F7	Allows the icon to be moved with keys.
Size	Alt+F8	Inactive (not an option for icons).
Minimize	Alt+F9	Inactive (not an option for icons).
Maximize	Alt+F10	Opens the associated window and makes it fill the screen.
Lower	Alt+F11	Moves icon to bottom of icon stack.
Close	Alt+F4	Removes client from **mwm** management.

Double-clicking button 1 on an icon normalizes the icon into its associated window. Double-clicking button 1 on the icon box's icon opens the icon box and allow access to the contained icons. (In general, double-clicking a mouse button offers a quick way to have a function performed. Another example is double-clicking button 1 with the pointer on the window menu button. This closes the window.)

Icon Box

When icons begin to clutter the screen, they can be packed into an "icon box." (To use an icon box, **mwm** must be started with the icon box configuration already set.) The icon box is a window manager window that holds client icons. Icons in the icon box can be manipulated with the mouse. The following table summarizes the behavior of this interface. Button actions apply whenever the pointer is on any part of the icon.

Button Action	Description
Button 1 click	Selects the icon.
Button 1 double click	Normalizes (opens) the associated window.
Button 1 double click	Raises an already *open* window to the top of the stack.
Button 1 drag	Moves the icon.

The window menu of the icon box differs from the window menu of a client window: The "Close" selection is replaced with the "PackIcons Alt+F12" selection. When selected, PackIcons packs the icons in the box to achieve neat rows with no empty slots.

Input Focus

mwm supports (by default) a keyboard input focus policy of explicit selection. This means when a window is selected to get keyboard input, it continues to get keyboard input until the window is withdrawn from window management, another window is explicitly selected to get keyboard input, or the window is iconified. There are numerous resources that control the input focus. The client window with the keyboard input focus has the active window appearance with a visually distinctive window frame.

The following tables summarize the keyboard input focus selection behavior:

Button Action	Object	Function Description
Button 1 press	Window / window frame	Keyboard focus selection
Button 1 press	Icon	Keyboard focus selection

Key Action	Function Description
[Alt][Tab]	Move input focus to next window in window stack.
[Alt][Shift][Tab]	Move input focus to previous window in window stack.

Window stacking

The stacking order of windows may be changed as a result of setting the keyboard input focus, iconifying a window, or by doing a window manager window stacking function.

When a window is iconified, the window's icon is placed on the bottom of the stack.

The following table summarizes the default window stacking behavior of the window manager:

Key Action	Function Description
[Alt][ESC]	Put bottom window on top of stack.
[Alt][Shift][ESC]	Put top window on bottom of stack.

A window can also be raised to the top when it gets the keyboard input focus (e.g., by doing a button 1 press on the window or by using [Alt][Tab]) if this auto-raise feature is enabled with the **focusAutoRaise** resource.

X DEFAULTS

mwm is configured from its resource database. This database is built from the following sources. They are listed in order of precedence, low to high:

app-defaults/Mwm
RESOURCE_MANAGER root window property or $HOME/.Xdefaults
XENVIRONMENT variable or $HOME/.Xdefaults-host
mwm command line options

Entries in the resource database may refer to other resource files for specific types of resources. These include files that contain bitmaps, fonts, and **mwm** specific resources such as menus and behavior specifications (i.e., button and key bindings).

Mwm is the resource class name of **mwm** and **mwm** is the resource name used by **mwm** to look up resources. In the following discussion of resource specification "Mwm" and "mwm" can be used interchangeably.

mwm uses the following types of resources:

Component Appearance Resources:

These resources specify appearance attributes of window manager user interface components. They can be applied to the appearance of window manager menus, feedback windows (e.g., the window reconfiguration feedback window), client window frames, and icons.

Specific Appearance and Behavior Resources:

These resources specify **mwm** appearance and behavior (e.g., window management policies). They are not set separately for different **mwm** user interface components.

Client Specific Resources:

These **mwm** resources can be set for a particular client window or class of client windows. They specify client-specific icon and client window frame appearance and behavior.

Resource identifiers can be either a resource name (e.g., foreground) or a resource class (e.g., Foreground). If the value of a resource is a filename and if the filename is prefixed by "~/", then it is relative to the path contained in the *$HOME* environment variable (generally the user's home directory). This is the only environment variable **mwm** uses directly ($XENVIRONMENT is used by the resource manager).

Component Appearance Resources

The syntax for specifying *component appearance resources* that apply to window manager icons, menus, and client window frames is

Mwm* *resource_id*

For example, **Mwm*foreground** is used to specify the foreground color for **mwm** menus, icons, and client window frames.

The syntax for specifying *component appearance resources* that apply to a particular **mwm** component is

Mwm*[menu|icon|client|feedback]* *resource_id*

If *menu* is specified, the resource is applied only to **mwm** menus; if *icon* is specified, the resource is applied to icons; and if *client* is specified, the resource is applied to client window frames. For example, **Mwm*icon*foreground** is used to specify the foreground color for **mwm** icons, **Mwm*menu*foreground** specifies the foreground color for **mwm** menus, and **Mwm*client*foreground** is used to specify the foreground color for **mwm** client window frames.

The appearance of the title area of a client window frame (including window management buttons) can be separately configured. The syntax for configuring the title area of a client window frame is:

Mwm*client*title* *resource_id*

For example, **Mwm*client*title*foreground** specifies the foreground color for the title area. Defaults for title area resources are based on the values of the corresponding client window frame resources.

The appearance of menus can be configured based on the name of the menu. The syntax for specifying menu appearance by name is:

Mwm*menu* *menu_name* ***resource_id***

For example, **Mwm*menu*my_menu*foreground** specifies the foreground color for the menu named **my_menu**.

The following *component appearance resources* that apply to all window manager parts can be specified:

Component Appearance Resources - All Window Manager Parts			
Name	**Class**	**Value Type**	**Default**
background	Background	color	varies*
backgroundPixmap	BackgroundPixmap	string**	varies*
bottomShadowColor	Foreground	color	varies*
bottomShadowPixmap	BottomShadowPixmap	string**	varies*
fontList	FontList	string***	"fixed"
foreground	Foreground	color	varies*
saveUnder	SaveUnder	T/F	F
topShadowColor	Background	color	varies*
topShadowPixmap	TopShadowPixmap	string**	varies*

*The default is chosen based on the visual type of the screen. **Pixmap image name. See XmInstallImage(3). ***X11 R3 Font description.

background (class **Background**)
This resource specifies the background color. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

backgroundPixmap (class **BackgroundPixmap**)
This resource specifies the background Pixmap of the **mwm** decoration when the window is inactive (does not have the keyboard focus). The default value is chosen based on the visual type of the screen.

bottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color. This color is used for the lower and right bevels of the window manager decoration. Any legal X color may be specified. The default value is chosen based on the visual type of the screen.

bottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap. This Pixmap is used for the lower and right bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

fontList (class **Font**)
This resource specifies the font used in the window manager decoration. The character encoding of the font should match the character encoding of the strings that are used. The default is "fixed."

foreground (class **Foreground**)
This resource specifies the foreground color. The default is chosen based on the visual type of the screen.

saveUnder (class **SaveUnder**)
This is used to indicate whether "save unders" are used for **mwm** components. For this to have any effect, save unders must be implemented by the X server. If save unders are

implemented, the X server will save the contents of windows obscured by windows that have the save under attribute set. If the saveUnder resource is True, **mwm** will set the save under attribute on the window manager frame of any client that has it set. If saveUnder is False, save unders will not be used on any window manager frames. The default value is False.

topShadowColor (class **Background**)
This resource specifies the top shadow color. This color is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

topShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap. This Pixmap is used for the upper and left bevels of the window manager decoration. The default is chosen based on the visual type of the screen.

The following *component appearance resources* that apply to frame and icons can be specified:

Frame and Icon Components			
Name	Class	Value Type	Default
activeBackground	Background	color	varies*
activeBackgroundPixmap	BackgroundPixmap	string**	varies*
activeBottomShadowColor	Foreground	color	varies*
activeBottomShadowPixmap	BottomShadowPixmap	string**	varies*
activeForeground	Foreground	color	varies*
activeTopShadowColor	Background	color	varies*
activeTopShadowPixmap	TopShadowPixmap	string**	varies*

*The default is chosen based on the visual type of the screen. **See XmInstallImage(3).

activeBackground (class **Background**)
This resource specifies the background color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBackgroundPixmap (class **ActiveBackgroundPixmap**)
This resource specifies the background Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the mwm decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeForeground (class **Foreground**)
This resource specifies the foreground color of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowColor (class **Background**)
This resource specifies the top shadow color of the **mwm** decoration when the window is

active (has the keyboard focus). The default is chosen based on the visual type of the screen.

activeTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the **mwm** decoration when the window is active (has the keyboard focus). The default is chosen based on the visual type of the screen.

Specific Appearance And Behavior Resources
The syntax for specifying *specific appearance and behavior resources* is

Mwm*resource_id*

For example, **Mwm*keyboardFocusPolicy** specifies the window manager policy for setting the keyboard focus to a particular client window.

The following *specific appearance and behavior resources* can be specified:

Specific Appearance and Behavior Resources			
Name	**Class**	**Value Type**	**Default**
autoKeyFocus	AutoKeyFocus	T/F	T
autoRaiseDelay	AutoRaiseDelay	millisec	500
bitmapDirectory	BitmapDirectory	directory	/usr/include/X11/bitmaps
buttonBindings	ButtonBindings	string	NULL
cleanText	CleanText	T/F	T
clientAutoPlace	ClientAutoPlace	T/F	T
colormapFocusPolicy	ColormapFocusPolicy	string	keyboard
configFile	ConfigFile	file	.mwmrc
deiconifyKeyFocus	DeiconifyKeyFocus	T/F	T
doubleClickTime	DoubleClickTime	millisec.	500
enforceKeyFocus	EnforceKeyFocus	T/F	T
fadeNormalIcon	FadeNormalIcon	T/F	F
frameBorderWidth	FrameBorderWidth	pixels	5
iconAutoPlace	IconAutoPlace	T/F	T
iconBoxGeometry	IconBoxGeometry	string	6x1+0-0
iconBoxName	IconBoxName	string	iconbox
iconBoxTitle	IconBoxTitle	string	Icons
iconClick	IconClick	T/F	T
iconDecoration	IconDecoration	string	varies
iconImageMaximum	IconImageMaximum	wxh	50x50
iconImageMinimum	IconImageMinimum	wxh	32x32
iconPlacement	IconPlacement	string	left bottom
iconPlacementMargin	IconPlacementMargin	pixels	varies
interactivePlacement	InteractivePlacement	T/F	F
keyBindings	KeyBindings	string	system
keyboardFocusPolicy	KeyboardFocusPolicy	string	explicit
limitResize	LimitResize	T/F	T
lowerOnIconify	LowerOnIconify	T/F	T
maximumMaximumSize	MaximumMaximumSize	wxh (pixels)	2X screen w&h
moveThreshold	MoveThreshold	pixels	4
passButtons	PassButtons	T/F	F
passSelectButton	PassSelectButton	T/F	T
positionIsFrame	PositionIsFrame	T/F	T
positionOnScreen	PositionOnScreen	T/F	T
quitTimeout	QuitTimeout	millisec.	1000
resizeBorderWidth	ResizeBorderWidth	pixels	10
resizeCursors	ResizeCursors	T/F	T
showFeedback	ShowFeedback	string	all
startupKeyFocus	StartupKeyFocus	T/F	T
transientDecoration	TransientDecoration	string	system title
transientFunctions	TransientFunctions	string	-minimize -maximize
useIconBox	UseIconBox	T/F	F
wMenuButtonClick	WMenuButtonClick	T/F	T

wMenuButtonClick2	WMenuButtonClick2	T/F	T

autoKeyFocus (class **AutoKeyFocus**)
This resource is only available when the keyboard input focus policy is explicit. If autoKeyFocus is given a value of True, then when a window with the keyboard input focus is withdrawn from window management or is iconified, the focus is set to the previous window that had the focus. If the value given is False, there is no automatic setting of the keyboard input focus. The default value is True.

autoRaiseDelay (class **AutoRaiseDelay**)
This resource is only available when the focusAutoRaise resource is True and the keyboard focus policy is pointer. The autoRaiseDelay resource specifies the amount of time (in milliseconds) that **mwm** will wait before raising a window after it gets the keyboard focus. The default value of this resource is 500 (ms).

bitmapDirectory (class **BitmapDirectory**)
This resource identifies a directory to be searched for bitmaps referenced by **mwm** resources. This directory is searched if a bitmap is specified without an absolute pathname. The default value for this resource is "/usr/include/X11/bitmaps".

buttonBindings (class **ButtonBindings**)
This resource identifies the set of button bindings for window management functions. The named set of button bindings is specified in the **mwm resource description file.** These button bindings are *merged* with the built-in default bindings. The default value for this resource is NULL (i.e., no button bindings are added to the built-in button bindings).

cleanText (class**CleanText**)
This resource controls the display of window manager text in the client title and feedback windows. If the default value of True is used, the text is drawn with a clear (no stipple) background. This makes text easier to read on monochrome systems where a backgroundPixmap is specified. Only the stippling in the area immediately around the text is cleared. If False, the text is drawn directly on top of the existing background.

clientAutoPlace (class **ClientAutoPlace**)
This resource determines the position of a window when the window has not been given a user specified position. With a value of True, windows are positioned with the top left corners of the frames offset horizontally and vertically. A value of False causes the currently configured position of the window to be used. In either case, **mwm** will attempt to place the windows totally on-screen. The default value is True.

colormapFocusPolicy (class **ColormapFocusPolicy**)
This resource indicates the colormap focus policy that is to be used. If the resource value is explicit then a colormap selection action is done on a client window to set the colormap focus to that window. If the value is pointer then the client window containing the pointer has the colormap focus. If the value is keyboard then the client window that has the keyboard input focus will have the colormap focus. The default value for this resource is keyboard.

configFile (class **ConfigFile**)
The resource value is the pathname for an **mwm resource description file.** The default is **.mwmrc** in the user's home directory (based on the $HOME environment variable) if this file exists, otherwise **/usr/lib/X11/system.mwmrc.**

deiconifyKeyFocus (class **DeiconifyKeyFocus**)
This resource only applies when the keyboard input focus policy is explicit. If a value of True is used, a window will receive the keyboard input focus when it is normalized (deiconified). True is the default value.

doubleClickTime (class **DoubleClickTime**)
This resource is used to set the maximum time (in ms) between the clicks (button presses) that make up a double-click. The default value of this resource is 500 (ms).

enforceKeyFocus (class **EnforceKeyFocus**)
If this resource is given a value of True, then the keyboard input focus is always explicitly set to selected windows even if there is an indication that they are "globally active" input windows. (An example of a globally active window is a scroll bar that can be operated without setting the focus to that client.) If the resource is False, the keyboard input focus is not explicitly set to globally active windows. The default value is True.

fadeNormalIcon (class **FadeNormalIcon**)
If this resource is given a value of True, an icon is grayed out whenever it has been normalized (its window has been opened). The default value is False.

frameBorderWidth (class **FrameBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border without resize handles. The border width includes the 3-D shadows. The default value is 5 pixels.

iconAutoPlace (class **IconAutoPlace**)
This resource indicates whether icons are automatically placed on the screen by **mwm**, or are placed by the user. Users may specify an initial icon position and may move icons after initial placement; however, **mwm** will adjust the user-specified position to fit into an invisible grid. When icons are automatically placed, **mwm** places them into the grid using a scheme set with the iconPlacement resource. If the iconAutoPlace resource has a value of True, then **mwm** does automatic icon placement. A value of False allows user placement. The default value of this resource is True.

iconBoxGeometry (class **IconBoxGeometry**)
This resource indicates the initial position and size of the icon box. The value of the resource is a standard window geometry string with the following syntax:

[=][*width*x*height*][{+-}*xoffset*{+-}*yoffset*]

If the offsets are not provided, the iconPlacement policy is used to determine the initial placement. The units for width and height are columns and rows.

The actual screen size of the icon box window will depend on the iconImageMaximum (size) and iconDecoration resources. The default value for size is (6 * iconWidth + padding) wide by (1 * iconHeight + padding) high. The default value of the location is +0 -0.

iconBoxName (class **IconBoxName**)
This resource specifies the name that is used to look up icon box resources. The default name is iconbox.

iconBoxTitle (class **IconBoxTitle**)
This resource specifies the name that is used in the title area of the icon box frame. The default value is Icons.

iconClick (class **IconClick**)
When this resource is given the value of True, the system menu is posted and left posted when an icon is clicked. The default value is True.

iconDecoration (class **IconDecoration**)
This resource specifies the general icon decoration. The resource value is label"(only the label part is displayed) or image (only the image part is displayed) or label image (both the label and image parts are displayed). A value of activelabel can also be specified to get a label (not truncated to the width of the icon) when the icon is selected. The default icon decoration for icon box icons is that each icon has a label part and an image part (label image). The default icon decoration for stand-alone icons is that each icon

has an active label part, a label part and an image part (activelabel label image).

iconImageMaximum (class **IconImageMaximum**)
This resource specifies the maximum size of the icon *image*. The resource value is *widthxheight* (e.g., 64x64). The maximum supported size is 128x128. The default value of this resource is 50x50.

iconImageMinimum (class **IconImageMinimum**)
This resource specifies the minimum size of the icon *image*. The resource value is *widthxheight* (e.g., 32x50). The minimum supported size is 16x16. The default value of this resource is 32x32.

iconPlacement (class **IconPlacement**)
This resource specifies the icon placement scheme to be used. The resource value has the following syntax:

primary_layout secondary_layout

The layout values are one of the following:

top	Lay the icons out top to bottom.
bottom	Lay the icons out bottom to top.
left	Lay the icons out left to right.
right	Lay the icons out right to left.

A horizontal (vertical) layout value should not be used for both the *primary_layout* and the *secondary_layout* (e.g., don't use top for the *primary_layout* and bottom for the *secondary_layout*). The *primary_layout* indicates whether, when an icon placement is done, the icon is placed in a row or a column and the direction of placement. The *secondary_layout* indicates where to place new rows or columns. For example, top right indicates that icons should be placed top to bottom on the screen and that columns should be added from right to left on the screen. The default placement is left bottom (icons are placed left to right on the screen, with the first row on the bottom of the screen, and new rows added from the bottom of the screen to the top of the screen).

iconPlacementMargin (class **IconPlacementMargin**)
This resource sets the distance between the edge of the screen and the icons that are placed along the edge of the screen. The value should be greater than or equal to 0. A default value (see below) is used if the value specified is invalid. The default value for this resource is equal to the space between icons as they are placed on the screen (this space is based on maximizing the number of icons in each row and column).

interactivePlacement (class **InteractivePlacement**)
This resource controls the initial placement of new windows on the screen. If the value is True, then the pointer shape changes before a new window is placed on the screen to indicate to the user that a position should be selected for the upper-left hand corner of the window. If the value is False, then windows are placed according to the initial window configuration attributes. The default value of this resource is False.

keyBindings (class **KeyBindings**)
This resource identifies the set of key bindings for window management functions. If specified these key bindings *replace* the built-in default bindings. The named set of key bindings is specified in **mwm resource description file**. The default value for this resource is the set of system-compatible key bindings.

keyboardFocusPolicy (class **KeyboardFocusPolicy**)
If set to pointer, the keyboard focus policy is to have the keyboard focus set to the client window that contains the pointer (the pointer could also be in the client window decoration that **mwm** adds). If set to explicit, the policy is to have the keyboard focus set to

a client window when the user presses button 1 with the pointer on the client window or any part of the associated **mwm** decoration. The default value for this resource is explicit.

limitResize (class **LimitResize**)
If this resource is True, the user is not allowed to resize a window to greater than the maximum size. The default value for this resource is True.

lowerOnIconify (class **LowerOnIconify**)
If this resource is given the default value of True, a window's icon appears on the bottom of the window stack when the window is minimized (iconified). A value of False places the icon in the stacking order at the same place as its associated window.

maximumMaximumSize (class **MaximumMaximumSize**)
This resource is used to limit the maximum size of a client window as set by the user or client. The resource value is *widthxheight* (e.g., 1024x1024) where the width and height are in pixels. The default value of this resource is twice the screen width and height.

moveThreshold (class **MoveThreshold**)
This resource is used to control the sensitivity of dragging operations that move windows and icons. The value of this resource is the number of pixels that the locator will be moved with a button down before the move operation is initiated. This is used to prevent window/icon movement when a click or double-click is done and there is unintentional pointer movement with the button down. The default value of this resource is 4 (pixels).

passButtons (class **PassButtons**)
This resource indicates whether or not button press events are passed to clients after they are used to do a window manager function in the client context. If the resource value is False, then the button press will not be passed to the client. If the value is True, the button press is passed to the client window. The window manager function is done in either case. The default value for this resource is False.

passSelectButton (class **PassSelectButton**)
This resource indicates whether or not the keyboard input focus selection button press (if keyboardFocusPolicy is explicit) is passed on to the client window or used to do a window management action associated with the window decorations. If the resource value is False then the button press will not be used for any operation other than selecting the window to be the keyboard input focus; if the value is True, the button press is passed to the client window or used to do a window management operation, if appropriate. The keyboard input focus selection is done in either case. The default value for this resource is True.

positionIsFrame (class **PositionIsFrame**)
This resource indicates how client window position information (from the WM_NORMAL_HINTS property and from configuration requests) is to be interpreted. If the resource value is True then the information is interpreted as the position of the **mwm** client window frame. If the value is False then it is interpreted as being the position of the client area of the window. The default value of this resource is True.

positionOnScreen (class **PositionOnScreen**)
This resource is used to indicate that windows should initially be placed (if possible) so that they are not clipped by the edge of the screen (if the resource value is True). If a window is larger then the size of the screen then at least the upper left corner of the window will be on-screen. If the resource value is False, then windows are placed in the requested position even if totally off-screen. The default value of this resource is True.

quitTimeout (class **QuitTimeout**)
This resource specifies the amount of time (in milliseconds) that **mwm** will wait for a

client to update the WM_COMMAND property after **mwm** has sent the WM_SAVE_YOURSELF message. This protocol will only be used for those clients that have a WM_SAVE_YOURSELF atom and no WM_DELETE_WINDOW atom in the WM_PROTOCOLS client window property. The default value of this resource is 1000 (ms). (Refer to the f.kill function for additional information.)

resizeBorderWidth (class **ResizeBorderWidth**)
This resource specifies the width (in pixels) of a client window frame border with resize handles. The specified border width includes the 3-D shadows. The default is 10 (pixels).

resizeCursors (class **ResizeCursors**)
This is used to indicate whether the resize cursors are always displayed when the pointer is in the window size border. If True the cursors are shown, otherwise the window manager cursor is shown. The default value is True.

showFeedback (class **ShowFeedback**)
This resource controls when feedback information is displayed. It controls both window position and size feedback during move or resize operations and initial client placement. It also controls window manager message and dialog boxes. The value for this resource is a list of names of the feedback options to be enabled; the names must be separated by a space. The names of the feedback options are shown below:

Name	Description
all	Show all feedback. (Default value.)
behavior	Confirm behavior switch.
move	Show position during move.
none	Show no feedback.
placement	Show position and size during initial placement.
resize	Show size during resize.
restart	Confirm **mwm** restart.

The following command line illustrates the syntax for showFeedback:

Mwm*showFeedback: placement resize behavior restart

This resource specification provides feedback for initial client placement and resize, and enables the dialog boxes to confirm the restart and set behavior functions. It disables feedback for the move function.

startupKeyFocus (class **StartupKeyFocus**)
This resource is only available when the keyboard input focus policy is explicit. When given the default value of True, a window gets the keyboard input focus when the window is mapped (i.e., initially managed by the window manager).

transientDecoration (class **TransientDecoration**)
This controls the amount of decoration that Mwm puts on transient windows. The decoration specification is exactly the same as for the **clientDecoration** (client specific) resource. Transient windows are identified by the WM_TRANSIENT_FOR property which is added by the client to indicate a relatively temporary window. The default value for this resource is menu title (i.e., transient windows will have resize borders and a titlebar with a window menu button).

transientFunctions (class **TransientFunctions**)
This resource is used to indicate which window management functions are applicable (or not applicable) to transient windows. The function specification is exactly the same as for the **clientFunctions** (client specific) resource. The default value for this resource is -minimize -maximize.

useIconBox (class **UseIconBox**)
If this resource is given a value of True, icons are placed in an icon box. When an icon box is not used, the icons are placed on the root window (default value).

wMenuButtonClick (class **WMenuButtonClick**)
This resource indicates whether a click of the mouse when the pointer is over the window menu button will post and leave posted the system menu. If the value given this resource is True, then the menu will remain posted. True is the default value for this resource.

wMenuButtonClick2 (class **WMenuButtonClick2**)
When this resource is given the default value of True, a double-click action on the window menu button will do an f.kill function.

Client Specific Resources

The syntax for specifying *client specific resources* is

Mwm*client_name_or_class***resource_id*

For example, **Mwm*mterm*windowMenu** is used to specify the window menu to be used with mterm clients.

The syntax for specifying *client specific resources* for all classes of clients is

Mwm*resource_id*

Specific client specifications take precedence over the specifications for all clients. For example, **Mwm*windowMenu** is used to specify the window menu to be used for all classes of clients that don't have a window menu specified.

The syntax for specifying resource values for windows that have an unknown name and class (i.e. the window does not have a WM_CLASS property associated with it) is

Mwm*defaults*resource_id*

For example, **Mwm*defaults*iconImage** is used to specify the icon image to be used for windows that have an unknown name and class.

The following *client specific resources* can be specified:

Client Specific Resources			
Name	**Class**	**Value Type**	**Default**
clientDecoration	ClientDecoration	string	all
clientFunctions	ClientFunctions	string	all
focusAutoRaise	FocusAutoRaise	T/F	T
iconImage	IconImage	pathname	(image)
iconImageBackground	Background	color	icon background
iconImageBottomShadowColor	Foreground	color	icon bottom shadow
iconImageBottomShadowPixmap	BottomShadowPixmap	color	icon bottom shadow pixmap
iconImageForeground	Foreground	color	icon foreground
iconImageTopShadowColor	Background	color	icon top shadow color
iconImageTopShadowPixmap	TopShadowPixmap	color	icon top shadow pixmap
matteBackground	Background	color	background
matteBottomShadowColor	Foreground	color	bottom shadow color
matteBottomShadowPixmap	BottomShadowPixmap	color	bottom shadow pixmap
matteForeground	Foreground	color	foreground
matteTopShadowColor	Background	color	top shadow color
matteTopShadowPixmap	TopShadowPixmap	color	top shadow pixmap
matteWidth	MatteWidth	pixels	0
maximumClientSize	MaximumClientSize	wxh	fill the screen
useClientIcon	UseClientIcon	T/F	F
windowMenu	WindowMenu	string	string

clientDecoration (class **ClientDecoration**)

This resource controls the amount of window frame decoration. The resource is specified as a list of decorations to specify their inclusion in the frame. If a decoration is preceded by a minus sign, then that decoration is excluded from the frame. The ***sign*** of the first item in the list determines the initial amount of decoration. If the sign of the first decoration is minus, then **mwm** assumes all decorations are present and starts subtracting from that set. If the sign of the first decoration is plus (or not specified), then **mwm** starts with no decoration and builds up a list from the resource.

Name	Description
all	Include all decorations (default value).
border	Window border.
maximize	Maximize button (includes title bar).
minimize	Minimize button (includes title bar).
none	No decorations.
resizeh	Border resize handles (includes border).
menu	Window menu button (includes title bar).
title	Title bar (includes border).

Examples:

Mwm*XClock.clientDecoration: -resizeh -maximize

This removes the resize handles and maximize button from XClock windows.

Mwm*XClock.clientDecoration: menu minimize border

This does the same thing as above. Note that either **menu** or **minimize** implies **title.**

clientFunctions (class **ClientFunctions**)

This resource is used to indicate which **mwm** functions are applicable (or not applicable) to the client window. The value for the resource is a list of functions. If the first function in the list has a minus sign in front of it, then **mwm** starts with all functions and subtracts from that set. If the first function in the list has a plus sign in front of it, then **mwm** starts with no functions and builds up a list. Each function in the list must be preceded by the appropriate plus or minus sign and be separated from the next function by a space.

The table below lists the functions available for this resource:

Name	Description
all	Include all functions (default value)
none	No functions
resize	f.resize
move	f.move
minimize	f.minimize
maximize	f.maximize
close	f.kill

focusAutoRaise (class **FocusAutoRaise**)

When the value of this resource is True, clients are made completely unobscured when they get the keyboard input focus. If the value is False, the stacking of windows on the display is not changed when a window gets the keyboard input focus. The default value is True.

iconImage (class **IconImage**)

This resource can be used to specify an icon image for a client (e.g., "Mwm*myclock*iconImage"). The resource value is a pathname for a bitmap file. The value of the (client specific) useClientIcon resource is used to determine whether or not user supplied icon images are used instead of client supplied icon images. The default value is to display a built-in window manager icon image.

iconImageBackground (class **Background**)

This resource specifies the background color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon background color (i.e., specified by "Mwm*background or Mwm*icon*background).

iconImageBottomShadowColor (class **Foreground**)

This resource specifies the bottom shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow color (i.e., specified by Mwm*icon*bottomShadowColor).

iconImageBottomShadowPixmap (class **BottomShadowPixmap**)

This resource specifies the bottom shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon bottom shadow Pixmap (i.e., specified by Mwm*icon*bottomShadowPixmap).

iconImageForeground (class **Foreground**)

This resource specifies the foreground color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon foreground color (i.e., specified by "Mwm*foreground or Mwm*icon*foreground).

iconImageTopShadowColor (class **Background**)

This resource specifies the top shadow color of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow color (i.e., specified by Mwm*icon*topShadowColor).

iconImageTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the icon image that is displayed in the image part of an icon. The default value of this resource is the icon top shadow Pixmap (i.e., specified by Mwm*icon*topShadowPixmap).

matteBackground (class **Background**)
This resource specifies the background color of the matte, when **matteWidth** is positive. The default value of this resource is the client background color (i.e., specified by "Mwm*background or Mwm*client*background).

matteBottomShadowColor (class **Foreground**)
This resource specifies the bottom shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow color (i.e., specified by "Mwm*bottomShadowColor or Mwm*client*bottomShadowColor).

matteBottomShadowPixmap (class **BottomShadowPixmap**)
This resource specifies the bottom shadow Pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client bottom shadow Pixmap (i.e., specified by "Mwm*bottomShadowPixmap or Mwm*client*bottomShadowPixmap).

matteForeground (class **Foreground**)
This resource specifies the foreground color of the matte, when **matteWidth** is positive. The default value of this resource is the client foreground color (i.e., specified by "Mwm*foreground or Mwm*client*foreground).

matteTopShadowColor (class **Background**)
This resource specifies the top shadow color of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow color (i.e., specified by "Mwm*topShadowColor or Mwm*client*topShadowColor).

matteTopShadowPixmap (class **TopShadowPixmap**)
This resource specifies the top shadow Pixmap of the matte, when **matteWidth** is positive. The default value of this resource is the client top shadow Pixmap (i.e., specified by "Mwm*topShadowPixmap or Mwm*client*topShadowPixmap).

matteWidth (class **MatteWidth**)
This resource specifies the width of the optional matte. The default value is 0, which effectively disables the matte.

maximumClientSize (class **MaximumClientSize**)
This is a size specification that indicates the client size to be used when an application is maximized. The resource value is specified as *width*x*height*. The width and height are interpreted in the units that the client uses (e.g., for terminal emulators this is generally characters). If this resource is not specified then the maximum size from the WM_NORMAL_HINTS property is used if set. Otherwise the default value is the size where the client window with window management borders fills the screen. When the maximum client size is not determined by the maximumClientSize resource, the maximumMaximumSize resource value is used as a constraint on the maximum size.

useClientIcon (class **UseClientIcon**)
If the value given for this resource is True, then a client supplied icon image will take precedence over a user supplied icon image. The default value is False, making the user supplied icon image have higher precedence than the client supplied icon image.

windowMenu (class **WindowMenu**)
This resource indicates the name of the menu pane that is posted when the window

menu is popped up (usually by pressing button 1 on the window menu button on the client window frame). Menu panes are specified in the **mwm resource description file** file. Window menus can be customized on a client class basis by specifying resources of the form **Mwm*** *client_name_or_class****windowMenu** (See "Mwm Resource Description File Syntax"). The default value of this resource is the name of the built-in window menu specification.

RESOURCE DESCRIPTION FILE

The **mwm resource description file** is a supplementary resource file that contains resource descriptions that are referred to by entries in the defaults files (.Xdefaults, app-defaults/Mwm). It contains descriptions of resources that are to be used by **mwm**, and that cannot be easily encoded in the defaults files (a bitmap file is an analogous type of resource description file). A particular **mwm resource description file** can be selected using the **configFile** resource.

The following types of resources can be described in the **mwm resource description file**:

Buttons	Window manager functions can be bound (associated) with button events.
Keys	Window manager functions can be bound (associated) with key press events.
Menus	Menu panes can be used for the window menu and other menus posted with key bindings and button bindings.

mwm Resource Description File Syntax

The **mwm resource description file** is a standard text file that contains items of information separated by blanks, tabs, and new lines characters. Blank lines are ignored. Items or characters can be quoted to avoid special interpretation (e.g., the comment character can be quoted to prevent it from being interpreted as the comment character). A quoted item can be contained in double quotes ("). Single characters can be quoted by preceding them by the back-slash character (\). All text from an unquoted # to the end of the line is regarded as a comment and is not interpreted as part of a resource description. If ! is the first character in a line, the line is regarded as a comment. Window manager functions can be accessed with button and key bindings, and with window manager menus. Functions are indicated as part of the specifications for button and key binding sets, and menu panes. The function specification has the following syntax:

function = *function_name* [*function_args*]
function_name = *window manager function*
function_args = {*quoted_item* | *unquoted_item*}

The following functions are supported. If a function is specified that isn't one of the supported functions then it is interpreted by **mwm** as *f.nop*.

f.beep This function causes a beep.

f.circle_down [**icon** | **window**]
This function causes the window or icon that is on the top of the window stack to be put on the bottom of the window stack (so that it is no longer obscuring any other window or icon). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (i.e. transient windows) are restacked with their associated primary window. Secondary windows always stay on top of the associated primary window and there can be no other primary windows between the secondary windows and their primary window. If an **icon** function argument is specified, then the function applies only to icons. If a **window** function argument is specified then the function applies only to windows.

f.circle_up [**icon** | **window**]

This function raises the window or icon on the bottom of the window stack (so that it is not obscured by any other windows). This function affects only those windows and icons that are obscuring other windows and icons, or that are obscured by other windows and icons. Secondary windows (i.e. transient windows) are restacked with their associated primary window. If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.exec or !

This function causes *command* to be executed (using the value of the $*SHELL* environment variable if it is set, otherwise */bin/sh*). The ! notation can be used in place of the **f.exec** function name.

f.focus_color

This function sets the colormap focus to a client window. If this function is done in a root context, then the default colormap (setup by the *X Window System* for the screen where **mwm** is running) is installed and there is no specific client window colormap focus. This function is treated as *f.nop* if *colormapFocusPolicy* is not explicit.

f.focus_key

This function sets the keyboard input focus to a client window or icon. This function is treated as *f.nop* if *keyboardFocusPolicy* is not explicit or the function is executed in a root context.

f.kill If the WM_DELETE_WINDOW protocol is set up, the client is sent a client message event indicating that the client window should be deleted. If the WM_SAVE_YOURSELF protocol is set up and the WM_DELETE_WINDOW protocol is not set up, the client is sent a client message event indicating that the client needs to prepare to be terminated. If the client does not have the WM_DELETE_WINDOW or WM_SAVE_YOURSELF protocol set up, this function causes a client's X connection to be terminated (usually resulting in termination of the client). Refer to the description of the *quitTimeout* resource and the *WM_PROTOCOLS* property.

f.lower [*-client*]

This function lowers a client window to the bottom of the window stack (where it obscures no other window). Secondary windows (i.e. transient windows) are restacked with their associated primary window. The *client* argument indicates the name or class of a client to lower. If the *client* argument is not specified then the context that the function was invoked in indicates the window or icon to lower.

f.maximize

This function causes a client window to be displayed with its maximum size.

f.menu

This function associates a cascading (pull-right) menu with a menu pane entry or a menu with a button or key binding. The *menu_name* function argument identifies the menu to be used.

f.minimize

This function causes a client window to be minimized (iconified). When a window is minimized when no icon box is used, its icon is placed on the bottom of the window stack (such that it obscures no other window). If an icon box is used, then the client's icon changes to its iconified form inside the icon box. Secondary windows (i.e. transient windows) are minimized with their associated primary window. There is only one icon for a primary window and all its secondary windows.

f.move

This function allows a client window to be interactively moved.

f.next_cmap
This function installs the next colormap in the list of colormaps for the window with the colormap focus.

f.next_key [icon | window | transient]
This function sets the keyboard input focus to the next window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as *f.nop* if *keyboardFocusPolicy* is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the **transient** argument is specified, then transient (secondary) windows are traversed (otherwise, if only **window** is specified, traversal is done only to the last focused window in a transient group). If an **icon** function argument is specified, then the function applies only to icons. If a **window** function argument is specified, then the function applies only to windows.

f.nop This function does nothing.

f.normalize
This function causes a client window to be displayed with its normal size. Secondary windows (i.e. transient windows) are placed in their normal state along with their associated primary window.

f.pack_icons
This function is used to relayout icons (based on the layout policy being used) on the root window or in the icon box. In general this causes icons to be "packed" into the icon grid.

f.pass_keys
This function is used to enable/disable (toggle) processing of key bindings for window manager functions. When it disables key binding processing all keys are passed on to the window with the keyboard input focus and no window manager functions are invoked. If the *f.pass_keys* function is invoked with a key binding to disable key binding processing the same key binding can be used to enable key binding processing.

f.post_wmenu
This function is used to post the window menu. If a key is used to post the window menu and a window menu button is present, the window menu is automatically placed with its top-left corner at the bottom-left corner of the window menu button for the client window. If no window menu button is present, the window menu is placed at the top-left corner of the client window.

f.prev_cmap
This function installs the previous colormap in the list of colormaps for the window with the colormap focus.

f.prev_key [icon | window | transient]
This function sets the keyboard input focus to the previous window/icon in the set of windows/icons managed by the window manager (the ordering of this set is based on the stacking of windows on the screen). This function is treated as *f.nop* if *keyboardFocusPolicy* is not explicit. The keyboard input focus is only moved to windows that do not have an associated secondary window that is application modal. If the *transient* argument is specified, then transient (secondary) windows are traversed (otherwise, if only *window* is specified, traversal is done only to the last focused window in a transient group). If an *icon* function argument is specified then the function applies only to icons. If an *window* function argument is specified then the function applies only to windows.

f.quit_mwm
This function terminates **mwm** (but NOT the X window system).

f.raise [-*client*]

This function raises a client window to the top of the window stack (where it is obscured by no other window). Secondary windows (i.e. transient windows) are restacked with their associated primary window. The *client* argument indicates the name or class of a client to raise. If the *client* argument is not specified then the context that the function was invoked in indicates the window or icon to raise.

f.raise_lower
This function raises a client window to the top of the window stack if it is partially obscured by another window, otherwise it lowers the window to the bottom of the window stack. Secondary windows (i.e. transient windows) are restacked with their associated primary window.

f.refresh
This function causes all windows to be redrawn.

f.refresh_win
This function causes a client window to be redrawn.

f.resize
This function allows a client window to be interactively resized.

f.restart
This function causes **mwm** to be restarted (effectively terminated and re-executed).

f.send_msg *message_number*
This function sends a client message of the type _MOTIF_WM_MESSAGES with the *message_type* indicated by the *message_number* function argument. The client message will only be sent if *message_number* is included in the client's _MOTIF_WM_MESSAGES property. A menu item label is grayed out if the menu item is used to do *f.send_msg* of a message that is not included in the client's _MOTIF_WM_MESSAGES property.

f.separator
This function causes a menu separator to be put in the menu pane at the specified location (the label is ignored).

f.set_behavior
This function causes the window manager to restart with the default OSF behavior (if a custom behavior is configured) or a custom behavior (if an OSF default behavior is configured).

f.title This function inserts a title in the menu pane at the specified location.

Each function may be constrained as to which resource types can specify the function (e.g., menu pane) and also what context the function can be used in (e.g., the function is done to the selected client window). Function contexts are

root No client window or icon has been selected as an object for the function.

window A client window has been selected as an object for the function. This includes the window's title bar and frame. Some functions are applied only when the window is in its normalized state (e.g., f.maximize) or its maximized state (e.g., f.normalize).

icon An icon has been selected as an object for the function.

If a function is specified in a type of resource where it is not supported or is invoked in a context that does not apply then the function is treated as *f.nop*. The following table indicates the resource types and function contexts in which window manager functions apply.

Function	Contexts	Resources
f.beep	root,icon,window	button,key,menu

f.circle_down	root,icon,window	button,key,menu
f.circle_up	root,icon,window	button,key,menu
f.exec	root,icon,window	button,key,menu
f.focus_color	root,icon,window	button,key,menu
f.focus_key	root,icon,window	button,key,menu
f.kill	icon,window	button,key,menu
f.lower	root,icon,window	button,key,menu
f.maximize	icon,window(normal)	button,key,menu
f.menu	root,icon,window	button,key,menu
f.minimize	window	button,key,menu
f.move	icon,window	button,key,menu
f.next_cmap	root,icon,window	button,key,menu
f.next_key	root,icon,window	button,key,menu
f.nop	root,icon,window	button,key,menu
f.normalize	icon,window(maximized)	button,key,menu
f.pack_icons	root,icon,window	button,key,menu
f.pass_keys	root,icon,window	button,key,menu
f.post_wmenu	root,icon,window	button,key
f.prev_cmap	root,icon,window	button,key,menu
f.prev_key	root,icon,window	button,key,menu
f.quit_mwm	root	button,key,menu
f.raise	root,icon,window	button,key,menu
f.raise_lower	icon,window	button,key,menu
f.refresh	root,icon,window	button,key,menu
f.refresh_win	window	button,key,menu
f.resize	window	button,key,menu
f.restart	root	button,key,menu
f.send_msg	icon,window	button,key,menu
f.separator	root,icon,window	menu
f.set_behavior	root,icon,window	button,key,menu
f.title	root,icon,window	menu

Window Manager Event Specification

Events are indicated as part of the specifications for button and key binding sets, and menu panes.

Button events have the following syntax:

button = [*modifier_list*]<*button_event_name*>
modifier_list = *modifier_name* {*modifier_name*}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the button event occurs). The following table indicates the values that can be used for *modifier_name*. The [Alt] key is frequently labeled [Extend] or [Meta]. Alt and Meta can be used interchangeably in event specification.

Modifier	*Description*
Ctrl	Control Key
Shift	Shift Key
Alt	Alt/Meta Key
Meta	Meta/Alt Key
Lock	Lock Key
Mod1	Modifier1
Mod2	Modifier2
Mod3	Modifier3
Mod4	Modifier4
Mod5	Modifier5

The following table indicates the values that can be used for *button_event_name*.

Button	*Description*
Btn1Down	Button 1 Press
Btn1Up	Button 1 Release
Btn1Click	Button 1 Press and Release
Btn1Click2	Button 1 Double Click
Btn2Down	Button 2 Press
Btn2Up	Button 2 Release
Btn2Click	Button 2 Press and Release
Btn2Click2	Button 2 Double Click
Btn3Down	Button 3 Press
Btn3Up	Button 3 Release
Btn3Click	Button 3 Press and Release
Btn3Click2	Button 3 Double Click
Btn4Down	Button 4 Press
Btn4Up	Button 4 Release
Btn4Click	Button 4 Press and Release
Btn4Click2	Button 4 Double Click
Btn5Down	Button 5 Press
Btn5Up	Button 5 Release
Btn5Click	Button 5 Press and Release
Btn5Click2	Button 5 Double Click

Key events that are used by the window manager for menu mnemonics and for binding to window manager functions are single key presses; key releases are ignored. Key events have the following syntax:

key = [*modifier_list*]<**Key**>*key_name*
modifier_list = *modifier_name* {*modifier_name*}

All modifiers specified are interpreted as being exclusive (this means that only the specified modifiers can be present when the key event occurs). Modifiers for keys are the same as those that apply to buttons. The *key_name* is an X11 keysym name. Keysym names can be found in the keysymdef.h file (remove the *XK_* prefix).

Button Bindings

The **buttonBindings** resource value is the name of a set of button bindings that are used to configure window manager behavior. A window manager function can be done when a button press occurs with the pointer over a framed client window, an icon or the root window. The context for indicating where the button press applies is also the context for invoking the window manager function when the button press is done (significant for functions that are context sensitive).

The button binding syntax is

Buttons *bindings_set_name*
{
button *context* *function*
button *context* *function*
.
.
button *context* *function*
}

The syntax for the *context* specification is

context = *object*[|*context*]
object = **root** | **icon** | **window** | **title** | **frame** | **border** | **app**

The context specification indicates where the pointer must be for the button binding to be effective. For example, a context of **window** indicates that the pointer must be over a client window or window management frame for the button binding to be effective. The **frame** context is for the window management frame around a client window (including the border and titlebar), the **border** context is for the border part of the window management frame (not including the titlebar), the **title** context is for the title area of the window management frame, and the **app** context is for the application window (not including the window management frame).

If an *f.nop* function is specified for a button binding, the button binding will not be done.

Key Bindings

The **keyBindings** resource value is the name of a set of key bindings that are used to configure window manager behavior. A window manager function can be done when a particular key is pressed. The context in which the key binding applies is indicated in the key binding specification. The valid contexts are the same as those that apply to button bindings.

The key binding syntax is

Keys *bindings_set_name*
{
key *context* *function*
key *context* *function*
.
.
key *context* *function*
}

If an *f.nop* function is specified for a key binding, the key binding will not be done. If an *f.post_wmenu* or *f.menu* function is bound to a key, **mwm** will automatically use the same key for removing the menu from the screen after it has been popped up.

The *context* specification syntax is the same as for button bindings. For key bindings, the **frame**, **title**, **border**, and **app** contexts are equivalent to the **window** context. The context for a key event is the window or icon that has the keyboard input focus (**root** if no window or icon has the keyboard input focus).

Menu Panes

Menus can be popped up using the *f.post_wmenu* and *f.menu* window manager functions. The context for window manager functions that are done from a menu is *root*, *icon* or *window* depending on how the menu was popped up. In the case of the *window* menu or menus popped up with

a key binding, the location of the keyboard input focus indicates the context. For menus popped up using a button binding, the context of the button binding is the context of the menu.

The menu pane specification syntax is

```
Menu menu_name
{
   label  [mnemonic]  [accelerator]  function
   label  [mnemonic]  [accelerator]  function
            .
            .
   label  [mnemonic]  [accelerator]  function
}
```

Each line in the *Menu* specification identifies the label for a menu item and the function to be done if the menu item is selected. Optionally a menu button mnemonic and a menu button keyboard accelerator may be specified. Mnemonics are functional only when the menu is posted and keyboard traversal applies.

The *label* may be a string or a bitmap file. The label specification has the following syntax:

```
label =          text | bitmap_file
bitmap_file =    @file_name
text =           quoted_item | unquoted_item
```

The string encoding for labels must be compatible with the menu font that is used. Labels are greyed out for menu items that do the *f.nop* function or an invalid function or a function that doesn't apply in the current context.

A *mnemonic* specification has the following syntax

```
mnemonic =    _character
```

The first matching *character* in the label is underlined. If there is no matching *character* in the label, no mnemonic is registered with the window manager for that label. Although the *character* must exactly match a character in the label, the mnemonic will not execute if any modifier (such as Shift) is pressed with the character key.

The *accelerator* specification is a key event specification with the same syntax as is used for key bindings to window manager functions.

ENVIRONMENT

mwm uses the environment variable **$HOME** specifying the user's home directory.

FILES

/usr/lib/X11/system.mwmrc
/usr/lib/X11/app-defaults/Mwm
$HOME/.Xdefaults
$HOME/.mwmrc

ORIGIN

HP

RELATED INFORMATION

X(1)
VendorShell(3)
XmInstallImage(3)

NOTE

mwm is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

oclock – display time of day

SYNOPSIS

oclock [-option ...]

DESCRIPTION

Clock simply displays the current time on an analog display

OPTIONS

-fg *foreground color*
choose a different color for the both hands and the jewel of the clock

-bg *background color*
choose a different color for the background.

-jewel *jewel color*
choose a different color for the jewel on the clock.

-minute *minute color*
choose a different color for the minute hand of the clock.

-hour *hour color*
choose a different color for the hour hand of the clock.

-backing { *WhenMapped Always NotUseful* }
selects an appropriate level of backing store.

-geometry *geometry*
define the initial window geometry; see *X(1)*.

-display *display*
specify the display to use; see *X(1)*.

-bd *border color*
choose a different color for the window border.

-bw *border width*
choose a different width for the window border. As the Clock widget changes its border around quite a bit, this is most usefully set to zero.

-noshape
causes the clock to not reshape itself and ancestors to exactly fit the outline of the clock.

COLORS

Although the default colors for the Clock widget are black and white, the widget was designed in color; unfortunately, the toolkit makes specifying these colors in a device-independent manner difficult. If you want to see the correct colors, add the following lines to your resource file:

```
Clock*Background: grey
Clock*BorderColor: light blue
Clock*hour: yellow
Clock*jewel: yellow
Clock*minute: yellow
```

SEE ALSO

X(1), X Toolkit documentation

COPYRIGHT

AUTHOR

Keith Packard, MIT X Consortium

NOTE

oclock is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

plaid – paint some plaid-like patterns in an X window

SYNOPSIS

plaid [-option ...]

OPTIONS

–b enable backing store for the window

–fg *color*

This option specifies the color to use for the foreground of the window. The default is "white."

–bg *color*

This option specifies the color to use for the background of the window. The default is "black."

–bd *color*

This option specifies the color to use for the border of the window. The default is "white."

–bw *number*

This option specifies the width in pixels of the border surrounding 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

NOTE

plaid is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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.

AUTHOR

Don Bennett, HP Labs

NOTE

puzzle is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

resize - utility to set TERMCAP and terminal settings to current window size

SYNOPSIS

resize [-u] [-s [row col]]

DESCRIPTION

Resize prints a shell command for setting the TERM and TERMCAP environment variables to indicate the current size of *xterm* window from which the command is run. For this output to take effect, *resize* must either be evaluated as part of the command line (usually done with a shell alias or function) or else redirected to a file which can then be read in. From the C shell (usually known as */bin/csh*), the following alias could be defined in the user's *.cshrc*:

```
% alias rs 'set noglob; 'eval resize''
```

After resizing the window, the user would type:

```
% rs
```

Users of versions of the Bourne shell (usually known as */bin/sh*) that don't have command functions will need to send the output to a temporary file and the read it back in with the "." command:

```
$ resize >/tmp/out
$ . /tmp/out
```

OPTIONS

The following options may be used with *resize*:

-u This option indicates that Bourne shell commands should be generated even if the user's current shell isn't */bin/sh*.

-c This option indicates that C shell commands should be generated even if the user's current shell isn't */bin/csh*.

-s [*rows columns*]
This option indicates that that Sun console escape sequences will be used instead of the special *xterm* escape code. If *rows* and *columns* are given, *resize* will ask the *xterm* to resize itself. However, the window manager may choose to disallow the change.

FILES

/etc/termcap for the base termcap entry to modify.
~/.cshrc user's alias for the command.

SEE ALSO

csh(1), tset(1), xterm(1)

AUTHORS

Mark Vandevoorde (MIT-Athena), Edward Moy (Berkeley)

BUGS

The *-u* or *-c* must appear to the left of *-s* if both are specified.

There should be some global notion of display size; termcap and terminfo need to be rethought in the context of window systems. (Fixed in 4.3BSD, and Ultrix-32 1.2)

NAME

twm - Tab Window Manager for the X Window System

SYNTAX

twm [-display *dpy*] [-s] [-f *initfile*] [-v]

DESCRIPTION

Twm is a window manager for the X Window System. It provides titlebars, shaped windows, several forms of icon management, user-defined macro functions, click-to-type and pointer-driven keyboard focus, and user-specified key and pointer button bindings.

This program is usually started by the user's session manager or startup script. When used from *xdm(1)* or *xinit(1)* without a session manager, *twm* is frequently executed in the foreground as the last client. When run this way, exiting *twm* causes the session to be terminated (i.e. logged out).

By default, application windows are surrounded by a "frame" with a titlebar at the top and a special border around the window. The titlebar contains the window's name, a rectangle that is lit when the window is receiving keyboard input, and function boxes known as "titlebuttons" at the left and right edges of the titlebar.

Pressing pointer Button1 (usually the left-most button unless it has been changed with *xmodmap*) on a titlebutton will invoke the function associated with the button. In the default interface, windows are iconified by clicking (pressing and then immediately releasing) the left titlebutton (which looks like a small X). Conversely, windows are deiconified by clicking in the associated icon or entry in the icon manager (see description of the variable **ShowIconManager** and of the function **f.showiconmgr**).

Windows are resized by pressing the right titlebutton (which resembles group of nested squares), dragging the pointer over edge that is to be moved, and releasing the pointer when the outline of the window is the desired size. Similarly, windows are moved by pressing in the title or highlight region, dragging a window outline to the new location, and then releasing when the outline is in the desired position. Just clicking in the title or highlight region raises the window without moving it.

When new windows are created, *twm* will honor any size and location information requested by the user (usually through *-geometry* command line argument or resources for the individual applications). Otherwise, an outline of the window's default size, its titlebar, and lines dividing the window into a 3x3 grid that track the pointer are displayed. Clicking pointer Button1 will position the window at the current position and give it the default size. Pressing pointer Button2 (usually the middle pointer button) and dragging the outline will give the window its current position but allow the sides to be resized as described above. Clicking pointer Button3 (usually the right pointer button) will give the window its current position but attempt to make it long enough to touch the bottom the screen.

OPTIONS

Twm accepts the following command line options:

–display *dpy*
This option specifies the X server to use.

–s This option indicates that only the default screen (as specified by **–display** or by the **DISPLAY** environment variable) should be managed. By default, *twm* will attempt to manage all screens on the display.

–f *filename*
This option specifies the name of the startup file to use. By default, *twm* will look in the user's home directory for files named *.twmrc.num* (where *num* is a screen number) or *.twmrc*.

–v This option indicates that *twm* should print error messages whenever an unexpected X Error event is received. This can be useful when debugging applications but can be

distracting in regular use.

CUSTOMIZATION

Much of *twm*'s appearance and behavior can be controlled by providing a startup file in one of the following locations (searched in order for each screen being managed when *twm* begins):

$HOME/.twmrc.*screennumber*
The *screennumber* is a small positive number (e.g. 0, 1, etc.) representing the screen number (e.g. the last number in the DISPLAY environment variable *host:displaynum.screennum*) that would be used to contact that screen of the display. This is intended for displays with multiple screens of differing visual types.

$HOME/.twmrc
This is the usual name for an individual user's startup file.

/usr/lib/X11/twm/system.twmrc
If neither of the preceding files are found, *twm* will look in this file for a default configuration. This is often tailored by the site administrator to provide convenient menus or familiar bindings for novice users.

If no startup files are found, *twm* will use the built-in defaults described above. The only resource used by *twm* is *bitmapFilePath* for a colon-separated list of directories to search when looking for bitmap files (for more information, see the *Athena Widgets* manual and *xrdb(1)*).

Twm startup files are logically broken up into three types of specifications: *Variables*, *Bindings*, *Menus*. The *Variables* section must come first and is used to describe the fonts, colors, cursors, border widths, icon and window placement, highlighting, autoraising, layout of titles, warping, use of the icon manager. The *Bindings* section usually comes second and is used to specify the functions that should be to be invoked when keyboard and pointer buttons are pressed in windows, icons, titles, and frames. The *Menus* section gives any user-defined menus (containing functions to be invoked or commands to be executed).

Variable names and keywords are case-insensitive. Strings must be surrounded by double quote characters (e.g. "blue") and are case-sensitive. A pound sign (#) outside of a string causes the remainder of the line in which the character appears to be treated as a comment.

VARIABLES

Many of the aspects of *twm*'s user interface are controlled by variables that may be set in the user's startup file. Some of the options are enabled or disabled simply by the presence of a particular keyword. Other options require keywords, numbers, strings, or lists of all of these.

Lists are surrounded by braces and are usually separated by whitespace or a newline. For example:

```
AutoRaise { "emacs" "XTerm" "Xmh" }
```

or

```
AutoRaise
{
        "emacs"
        "XTerm"
        "Xmh"
}
```

When a variable containing a list of strings representing windows is searched (e.g. to determine whether or not to enable autoraise as shown above), a string is considered to match a window if it is a case-sensitive prefix for the window's name name (given by the WM_NAME window property), resource name or class name (both given by the WM_CLASS window property). The preceding example would enable autoraise on windows named "emacs" as well as any *xterm* (since they are of class "XTerm") or xmh windows (which are of class "Xmh").

String arguments that are interpreted as filenames (see the **Pixmaps**, **Cursors**, and **IconDirectory** below) will prepend the user's directory (specified by the **HOME** environment variable) if the first character is a tilde (~). If, instead, the first character is a colon (:), the name is assumed to refer to one of the internal bitmaps that are used to create the default titlebars symbols: **:xlogo** or **:iconify** (both refer to the X used for the iconify button), **:resize** (the nested squares used by the resize button), and **:question** (the question mark used for non-existent bitmap files).

The following variables may be specified at the top of a *twm* startup file. Lists of Window name prefix strings are indicated by *win-list*. Optional arguments are shown in square brackets:

AutoRaise { *win-list* }
This variable specifies a list of windows that should automatically be raised whenever the pointer enters the window. This action can be interactively enabled or disabled on individual windows using the function **f.autoraise**.

AutoRelativeResize
This variable indicates that dragging out a window size (either when initially sizing the window with pointer Button2 or when resizing it) should not wait until the pointer has crossed the window edges. Instead, moving the pointer automatically causes the nearest edge or edges to move by the same amount. This allows allows the resizing windows that extend off the edge of the screen. If the pointer is in the center of the window, or if the resize is begun by pressing a titlebutton, *twm* will still wait for the pointer to cross a window edge (to prevent accidents). This option is particularly useful for people who like the press-drag-release method of sweeping out window sizes.

BorderColor *string* [{ *wincolorlist* }]
This variable specifies the default color of the border to be placed around all non-iconified windows, and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* specifies a list of window and color name pairs for specifying particular border colors for different types of windows. For example:

```
BorderColor "gray50"
{
        "XTerm"         "red"
        "xmh"   "green"
}
```

The default is "black".

BorderTileBackground *string* [{ *wincolorlist* }]
This variable specifies the default background color in the gray pattern used in unhighlighted borders (only if **NoHighlight** hasn't been set), and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* allows per-window colors to be specified. The default is "black".

BorderTileForeground *string* [{ *wincolorlist* }]
This variable specifies the default foreground color in the gray pattern used in unhighlighted borders (only if **NoHighlight** hasn't been set), and may only be given within a **Color** or **Monochrome** list. The optional *wincolorlist* allows per-window colors to be specified. The default is "white".

BorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding all client window frames if **ClientBorderWidth** has not been specified. This value is also used to set the border size of windows created by *twm* (such as the icon manager). The default is 2.

ButtonIndent *pixels*
This variable specifies the amount by which titlebuttons should be indented on all sides. Positive values cause the buttons to be smaller than the window text and highlight area

so that they stand out. Setting this and the **TitleButtonBorderWidth** variables to 0 makes titlebuttons be as tall and wide as possible. The default is 1.

ClientBorderWidth
This variable indicates that border width of a window's frame should be set to the initial border width of the window, rather than to the value of **BorderWidth**.

Color { *colors-list* }
This variable specifies a list of color assignments to be made if the default display is capable of displaying more than simple black and white. The *colors-list* is made up of the following color variables and their values: **DefaultBackground**, **DefaultForeground**, **MenuBackground**, **MenuForeground**, **MenuTitleBackground**, **MenuTitleForeground**, and **MenuShadowColor**. The following color variables may also be given a list of window and color name pairs to allow per-window colors to be specified (see **BorderColor** for details): **BorderColor**, **IconManagerHighlight**, **BorderTitleBackground**, **BorderTitleForeground**, **TitleBackground**, **TitleForeground**, **IconBackground**, **IconForeground**, **IconBorderColor**, **IconManagerBackground**, and **IconManagerForeground**. For example:

```
Color
{
        MenuBackground          "gray50"
        MenuForeground          "blue"
        BorderColor             "red" { "XTerm" "yellow" }
        TitleForeground     "yellow"
        TitleBackground         "blue"
}
```

All of these color variables may also be specified for the **Monochrome** variable, allowing the same initialization file to be used on both color and monochrome displays.

ConstrainedMoveTime *milliseconds*
This variable specifies the length of time between button clicks needed to begin a constrained move operation. Double clicking within this amount of time when invoking **f.move** will cause the window only be moved in a horizontal or vertical direction. Setting this value to 0 will disable constrained moves. The default is 400 milliseconds.

Cursors { *cursor-list* }
This variable specifies the glyphs that *twm* should use for various pointer cursors. Each cursor may be defined either from the **cursor** font or from two bitmap files. Shapes from the **cursor** font may be specified directly as:

cursorname "*string*"

where *cursorname* is one of the cursor names listed below, and *string* is the name of a glyph as found in the file /usr/include/X11/cursorfont.h (without the "XC_" prefix). If the cursor is to be defined from bitmap files, the following syntax is used instead:

cursorname "*image*" "*mask*"

The *image* and *mask* strings specify the names of files containing the glyph image and mask in *bitmap(1)* form. The bitmap files are located in the same manner as icon bitmap files. The following example shows the default cursor definitions:

```
Cursors
{
        Frame           "top_left_arrow"
        Title           "top_left_arrow"
        Icon            "top_left_arrow"
```

```
        IconMgr         "top_left_arrow"
        Move            "fleur"
        Resize          "fleur"
        Menu            "sb_left_arrow"
        Button          "hand2"
        Wait            "watch"
        Select          "dot"
        Destroy "pirate"
}
```

DecorateTransients
This variable indicates that transient windows (those containing a WM_TRANSIENT_FOR property) should have titlebars. By default, transients are not reparented.

DefaultBackground *string*
This variable specifies the background color to be used for sizing and information windows. The default is "white".

DefaultForeground *string*
This variable specifies the foreground color to be used for sizing and information windows. The default is "black".

DontIconifyByUnmapping { *win-list* }
This variable specifies a list of windows that should not be iconified by simply unmapping the window (as would be the case if **IconifyByUnmapping** had been set). This is frequently used to force some windows to be treated as icons while other windows are handled by the icon manager.

DontMoveOff
This variable indicates that windows should not be allowed to be moved off the screen. It can be overridden by the **f.forcemove** function.

DontSqueezeTitle [{ *win-list* }]
This variable indicates that titlebars should not be squeezed to their minimum size as described under **SqueezeTitle** below. If the optional window list is supplied, only those windows will be prevented from being squeezed.

ForceIcons
This variable indicates that icon pixmaps specified in the **Icons** variable should override any client-supplied pixmaps.

FramePadding *pixels*
This variable specifies the distance between the titlebar decorations (the button and text) and the window frame. The default is 2 pixels.

IconBackground *string* [{ *win-list* }]
This variable specifies the background color of icons, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "white".

IconBorderColor *string* [{ *win-list* }]
This variable specifies the color of the border used for icon windows, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconBorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding icon windows. The

default is 2.

IconDirectory *string*
This variable specifies the directory that should be searched if if a bitmap file cannot be found in any of the directories in the **bitmapFilePath** resource.

IconFont *string*
This variable specifies the font to be used to display icon names within icons. The default is "8x13".

IconForeground *string* [{ *win-list* }]
This variable specifies the foreground color to be used when displaying icons, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconifyByUnmapping [{ *win-list* }]
This variable indicates that windows should be iconified by being unmapped without trying to map any icons. This assumes that the user is will remap the window through the icon manager, the **f.warpto** function, or the *TwmWindows* menu. If the optional *win-list* is provided, only those windows will be iconified by simply unmapping. Windows that have both this and the **IconManagerDontShow** options set may not be accessible if no binding to the *TwmWindows* menu is set in the user's startup file.

IconManagerBackground *string* [{ *win-list* }]
This variable specifies the background color to use for icon manager entries, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "white".

IconManagerDontShow [{ *win-list* }]
This variable indicates that the icon manager should not display any windows. If the optional *win-list* is given, only those windows will not be displayed. This variable is used to prevent windows that are rarely iconified (such as *xclock* or *xload*) from taking up space in the icon manager.

IconManagerFont *string*
This variable specifies the font to be used when displaying icon manager entries. The default is "8x13".

IconManagerForeground *string* [{ *win-list* }]
This variable specifies the foreground color to be used when displaying icon manager entries, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete description of the *win-list*. The default is "black".

IconManagerGeometry *string* [*columns*]
This variable specifies the geometry of the icon manager window. The *string* argument is standard geometry specification that indicates the initial full size of the icon manager. The icon manager window is then broken into *columns* pieces and scaled according to the number of entries in the icon manager. Extra entries are wrapped to form additional rows. The default number of columns is 1.

IconManagerHighlight *string* [{ *win-list* }]
This variable specifies the border color to be used when highlighting the icon manager entry that currently has the focus, and can only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. See the **BorderColor** variable for a complete

description of the *win-list*. The default is "black".

IconManagers { *iconmgr-list* }
This variable specifies a list of icon managers to create. Each item in the *iconmgr-list* has the following format:

"winname" ["iconname"] "geometry" columns

where *winname* is the name of the windows that should be put into this icon manager, *iconname* is the name of that icon manager window's icon, *geometry* is a standard geometry specification, and *columns* is the number of columns in this icon manager as described in **IconManagerGeometry**. For example:

```
IconManagers
{
        "XTerm"         "=300x5+800+5"      5
        "myhost"        "=400x5+100+5"      2
}
```

Clients whose name or class is "XTerm" will have an entry created in the "XTerm" icon manager. Clients whose name was "myhost" would be put into the "myhost" icon manager.

IconManagerShow { *win-list* }
This variable specifies a list of windows that should appear in the icon manager. When used in conjunction with the **IconManagerDontShow** variable, only the windows in this list will be shown in the icon manager.

IconRegion *geomstring vgrav hgrav gridwidth gridheight*
This variable specifies an area on the root window in which icons are placed if no specific icon location is provided by the client. The *geomstring* is a quoted string containing a standard geometry specification. If more than one **IconRegion** lines are given, icons will be put into the succeeding icon regions when the first is full. The *vgrav* argument should be either **North** or **South** and control and is used to control whether icons are first filled in from the top or bottom of the icon region. Similarly, the *hgrav* argument should be either **East** or **West** and is used to control whether icons should be filled in from left from the right. Icons are laid out within the region in a grid with cells *gridwidth* pixels wide and *gridheight* pixels high.

Icons { *win-list* }
This variable specifies a list of window names and the bitmap filenames that should be used as their icons. For example:

```
Icons
{
        "XTerm"         "xterm.icon"
        "xfd"           "xfd_icon"
}
```

Windows that match "XTerm" and would not be iconified by unmapping, and would try to use the icon bitmap in the file "xterm.icon". If **ForceIcons** is specified, this bitmap will be used even if the client has requested its own icon pixmap.

InterpolateMenuColors
This variable indicates that menu entry colors should be interpolated between entry

specified colors. In the example below:

```
Menu "mymenu"
{
        "Title"         ("black":"red")         f.title
        "entry1"                                f.nop
        "entry2"                                f.nop
        "entry3"        ("white":"green")       f.nop
        "entry4"                                f.nop
        "entry5"        ("red":"white")         f.nop
}
```

the foreground colors for "entry1" and "entry2" will be interpolated between black and white, and the background colors between red and green. Similarly, the foreground for "entry4" will be half-way between white and red, and the background will be half-way between green and white.

MakeTitle { *win-list* }
This variable specifies a list of windows on which a titlebar should be placed and is used to request titles on specific windows when **NoTitle** has been set.

MaxWindowSize *string*
This variable specifies a geometry in which the width and height give the maximum size for a given window. This is typically used to restrict windows to the size of the screen. The default is "30000x30000".

MenuBackground *string*
This variable specifies the background color used for menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

MenuFont *string*
This variable specifies the font to use when displaying menus. The default is "8x13".

MenuForeground *string*
This variable specifies the foreground color used for menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

MenuShadowColor *string*
This variable specifies the color of the shadow behind pull-down menus and can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

MenuTitleBackground *string*
This variable specifies the background color for **f.title** entries in menus, and can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

MenuTitleForeground *string*
This variable specifies the foreground color for **f.title** entries in menus and can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

Monochrome { *colors* }
This variable specifies a list of color assignments that should be made if the screen has a depth of 1. See the description of **Colors**.

MoveDelta *pixels*
This variable specifies the number of pixels the pointer must move before the **f.move** function starts working. Also see the **f.deltastop** function. The default is zero pixels.

NoBackingStore
This variable indicates that *twm*'s menus should not request backing store to minimize repainting of menus. This is typically used with servers that can repaint faster than they can handle backing store.

NoCaseSensitive
This variable indicates that case should be ignored when sorting icon names in an icon manager. This option is typically used with applications that capitalize the first letter of their icon name.

NoDefaults
This variable indicates that *twm* should not supply the default titlebuttons and bindings. This option should only be used if the startup file contains a completely new set of bindings and definitions.

NoGrabServer
This variable indicates that *twm* should not grab the server when popping up menus and moving opaque windows.

NoHighlight [{ *win-list* }]
This variable indicates that borders should not be highlighted to track the location of the pointer. If the optional *win-list* is given, highlighting will only be disabled for those windows. When the border is highlighted, it will be drawn in the current **BorderColor**. When the border is not highlighted, it will be stippled with an gray pattern using the current **BorderTileForeground** and **BorderTileBackground** colors.

NoIconManagers
This variable indicates that no icon manager should be created.

NoMenuShadows
This variable indicates that menus should not have drop shadows drawn behind them. This is typically used with slower servers since it speeds up menu drawing at the expense of making the menu slightly harder to read.

NoRaiseOnDeiconify
This variable indicates that windows that are deiconified should not be raised.

NoRaiseOnMove
This variable indicates that windows should not be raised when moved. This is typically used to allow windows to slide underneath each other.

NoRaiseOnResize
This variable indicates that windows should not be raised when resized. This is typically used to allow windows to be resized underneath each other.

NoRaiseOnWarp
This variable indicates that windows should not be raised when the pointer is warped into them with the **f.warpto** function. If this option is set, warping to an occluded window may result in the pointer ending up in the occluding window instead the desired window (which causes unexpected behavior with **f.warpring**).

NoSaveUnders
This variable indicates that menus should not request save-unders to minimize window repainting following menu selection. It is typically used with displays that can repaint faster than they can handle save-unders.

NoStackMode [{ *win-list* }]
This variable indicates that client window requests to change stacking order should be ignored. If the optional *win-list* is given, only requests on those windows will be ignored. This is typically used to prevent applications from relentlessly popping themselves to the front of the window stack.

NoTitle [{ *win-list* }]
This variable indicates that windows should not have titlebars. If the optional *win-list* is given, only those windows will not have titlebars. **MakeTitle** may be used with this option to force titlebars to be put on specific windows.

NoTitleFocus

This variable indicates that *twm* should not set keyboard input focus to each window as it is entered. Normally, *twm* sets the focus so that focus and key events from the titlebar and icon managers are delivered to the application. If the pointer is moved quickly and *twm* is slow to respond, input can be directed to the old window instead of the new. This option is typically used to prevent this "input lag" and to work around bugs in older applications that have problems with focus events.

NoTitleHighlight [{ *win-list* }]

This variable indicates that the highlight area of the titlebar, which is used to indicate the window that currently has the input focus, should not be displayed. If the optional *win-list* is given, only those windows will not have highlight areas. This and the **SqueezeTitle** options can be set to substantially reduce the amount of screen space required by titlebars.

OpaqueMove

This variable indicates that the **f.move** function should actually move the window instead of just an outline so that the user can immediately see what the window will look like in the new position. This option is typically used on fast displays (particularly if **NoGrabServer** is set).

Pixmaps { *pixmaps* }

This variable specifies a list of pixmaps that define the appearance of various images. Each entry is a keyword indicating the pixmap to set, followed by a string giving the name of the bitmap file. The following pixmaps may be specified:

```
Pixmaps
{
        TitleHighlight   "gray1"
}
```

The default for *TitleHighlight* is to use an even stipple pattern.

RandomPlacement

This variable indicates that windows with no specified geometry should should be placed in a pseudo-random location instead of having the user drag out an outline.

ResizeFont *string*

This variable specifies the font to be used for in the dimensions window when resizing windows. The default is "fixed".

RestartPreviousState

This variable indicates that *twm* should attempt to use the WM_STATE property on client windows to tell which windows should be iconified and which should be left visible. This is typically used to make try to regenerate the state that the screen was in before the previous window manager was shutdown.

ShowIconManager

This variable indicates that the icon manager window should be displayed when *twm* is started. It can always be brought up using the **f.showiconmgr** function.

SortIconManager

This variable indicates that entries in the icon manager should be sorted alphabetically rather than by simply appending new windows to the end.

SqueezeTitle [{ *squeeze-list* }]

This variable indicates that *twm* should attempt to use the SHAPE extension to make titlebars occupy only as much screen space as they need, rather than extending all the way across the top of the window. The optional *squeeze-list* may be used to control the location of the squeezed titlebar along the top of the window. It contains entries of the

form:

"name" justification num denom

where *name* is a window name, *justification* is either **left**, **center**, or **right**, and *num* and *denom* are numbers specifying a ratio giving the relative position about which the titlebar is justified. The ratio is measured from left to right if the numerator is positive, and right to left if negative. A denominator of 0 indicates that the numerator should be measured in pixels. For convenience, the ratio 0/0 is the same as 1/2 for **center** and -1/1 for **right**. For example:

```
SqueezeTitle
{
        "XTerm"         left            0       0
        "xterm1"        left            1       3
        "xterm2"        left            2       3
        "oclock"        center          0       0
        "emacs"         right           0       0
}
```

The **DontSqueezeTitle** list can be used to turn off squeezing on certain titles.

StartIconified [{ *win-list* }]
This variable indicates that client windows should initially be left as icons until explicitly deiconified by the user. If the optional *win-list* is given, only those windows will be started iconic. This is useful for programs that do not support an *-iconic* command line option or resource.

TitleBackground *string* [{ *win-list* }]
This variable specifies the background color used in titlebars, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. The default is "white".

TitleButtonBorderWidth *pixels*
This variable specifies the width in pixels of the border surrounding titlebuttons. This is typically set to 0 to allow titlebuttons to take up as much space as possible and to not have a border. The default is 1.

TitleFont *string*
This variable specifies the font to used for displaying window names in titlebars. The default is "8x13".

TitleForeground *string* [{ *win-list* }]
This variable specifies the foreground color used in titlebars, and may only be specified inside of a **Color** or **Monochrome** list. The optional *win-list* is a list of window names and colors so that per-window colors may be specified. The default is "black".

TitlePadding *pixels*
This variable specifies the distance between the various buttons, text, and highlight areas in the titlebar. The default is 8 pixels.

UnknownIcon *string*
This variable specifies the filename of a bitmap file to be used as the default icon. This bitmap will be used as the icon of all clients which do not provide an icon bitmap and are not listed in the **Icons** list.

UsePPosition *string*
This variable specifies whether or not *twm* should honor program-requested locations (given by the **PPosition** flag in the WM_NORMAL_HINTS property) in the absence of a user-specified position. The argument *string* may have one of three values: **"off"** (the

default) indicating that *twm* should ignore the program-supplied position, **"on"** indicating that the position should be used, and **"non-zero"** indicating that the position should used if it is other than (0,0). The latter option is for working around a bug in older toolkits.

WarpCursor [{ *win-list* }]
This variable indicates that the pointer should be warped into windows when they are deiconified. If the optional *win-list* is given, the pointer will only be warped when those windows are deiconified.

WindowRing { *win-list* }
This variable specifies a list of windows along which the **f.warpring** function cycles.

WarpUnmapped
This variable indicates that that the **f.warpto** function should deiconify any iconified windows it encounters. This is typically used to make a key binding that will pop a particular window (such as *xmh*), no matter where it is. The default is for **f.warpto** to ignore iconified windows.

XorValue *number*
This variable specifies the value to use when drawing window outlines for moving and resizing. This should be set to a value that will result in a variety of of distinguishable colors when exclusive-or'ed with the contents of the user's typical screen. Setting this variable to 1 often gives nice results if adjacent colors in the default colormap are distinct. By default, *twm* will attempt to cause temporary lines to appear at the opposite end of the colormap from the graphics.

Zoom [*count*]
This variable indicates that outlines suggesting movement of a window to and from its iconified state should be displayed whenever a window is iconified or deiconified. The optional *count* argument specifies the number of outlines to be drawn. The default count is 8.

The following variables must be set after the fonts have been assigned, so it is usually best to put them at the end of the variables or beginning of the bindings sections:

DefaultFunction *function*
This variable specifies the function to be executed when a key or button event is received for which no binding is provided. This is typically bound to **f.nop**, **f.beep**, or a menu containing window operations.

WindowFunction *function*
This variable specifies the function to execute when a window is selected from the **TwmWindows** menu. If this variable is not set, the window will be deiconified and raised.

BINDINGS

After the desired variables have been set, functions may be attached titlebuttons and key and pointer buttons. Titlebuttons may be added from the left or right side and appear in the titlebar from left-to-right according to the order in which they are specified. Key and pointer button bindings may be given in any order.

Titlebuttons specifications must include the name of the pixmap to use in the button box and the function to be invoked when a pointer button is pressed within them:

LeftTitleButton *"bitmapname"* = *function*

or

RightTitleButton "*bitmapname*" = *function*

The *bitmapname* may refer to one of the built-in bitmaps (which are scaled to match **TitleFont**) by using the appropriate colon-prefixed name described above.

Key and pointer button specifications must give the modifiers that must be pressed, over which parts of the screen the pointer must be, and what function is to be invoked. Keys are given as strings containing the appropriate keysym name; buttons are given as the keywords **Button1**-**Button5**:

"FP1" = *modlist* : *context* : *function*
Button1 = *modlist* : *context* : *function*

The *modlist* is any combination of the modifier names **shift**, **control**, **lock**, **meta**, **mod1**, **mod2**, **mod3**, **mod4**, or **mod1** (which may be abbreviated as **s**, **c**, **l**, **m**, **m1**, **m2**, **m3**, **m4**, **m5**, respectively) separated by a vertical bar (|). Similarly, the *context* is any combination of **window**, **title**, **icon**, **root**, **frame**, **iconmgr**, their first letters (**iconmgr** abbreviation is **m**), or **all**, separated by a vertical bar. The *function* is any of the **f.** keywords described below. For example, the default startup file contains the following bindings:

```
Button1=      : root          : f.menu "TwmWindows"
Button1= m    : window | icon : f.function "move-or-lower"
Button2= m    : window | icon : f.iconify
Button3= m    : window | icon : f.function "move-or-raise"
Button1=      : title         : f.function "move-or-raise"
Button2=      : title         : f.raiselower
Button1=      : icon          : f.function "move-or-iconify"
Button2=      : icon          : f.iconify
Button1=      : iconmgr       : f.iconify
Button2=      : iconmgr       : f.iconify
```

A user who wanted to be able to manipulate windows from the keyboard could use the following bindings:

```
"F1"          =     : all          : f.iconify
"F2"          =     : all          : f.raiselower
"F3"          =     : all          : f.warpring "next"
"F4"          =     : all          : f.warpto "xmh"
"F5"          =     : all          : f.warpto "emacs"
"F6"          =     : all          : f.colormap "next"
"F7"          =     : all          : f.colormap "default"
"F20"         =     : all          : f.warptoscreen "next"
"Left"        = m   : all          : f.backiconmgr
"Right" = m | s : all              : f.forwiconmgr
"Up"          = m   : all          : f.upiconmgr
"Down" = m | s : all               : f.downiconmgr
```

Twm provides many more window manipulation primitives than can be conveniently stored in a titlebar, menu, or set of key bindings. Although a small set of defaults are supplied (unless the **NoDefaults** is specified), most users will want to have their most common operations bound to key and button strokes. To do this, *twm* associates names with each of the primitives and provides *user-defined functions* for building higher level primitives and *menus* for interactively selecting among groups of functions.

User-defined functions contain the name by which they are referenced in calls to **f.function** and a

list of other functions to execute. For example:

```
Function "move-or-lower"      { f.move f.deltastop f.lower }
Function "move-or-raise"      { f.move f.deltastop f.raise }
Function "move-or-iconify"    { f.move f.deltastop f.iconify }
Function "restore-colormap"   { f.colormap "default" f.lower }
```

The function name must be used in **f.function** exactly as it appears in the function specification.

In the descriptions below, if the function is said to operate on the selected window, but is invoked from a root menu, the cursor will be changed to the **Select** cursor and the next window to receive a button press will be chosen:

! *string* This is an abbreviation for **f.exec** *string*.

f.autoraise
This function toggles whether or not the selected window is raised whenever entered by the pointer. See the description of the variable **AutoRaise**.

f.backiconmgr
This function warps the pointer to the previous column in the current icon manager, wrapping back to the previous row if necessary.

f.beep This function sounds the keyboard bell.

f.bottomzoom
This function is similar to the **f.fullzoom** function, but resizes the window to fill only the bottom half of the screen.

f.circledown
This function lowers the top-most window that occludes another window.

f.circleup
This function raises the bottom-most window that is occluded by another window.

f.colormap *string*
This function rotates the colormaps (obtained from the WM_COLORMAP_WINDOWS property on the window) that *twm* will display when the pointer is in this window. The argument *string* may have one of the following values: **"next"**, **"prev"**, and **"default"**.

f.deiconify
This function deiconifies the selected window. If the window is not an icon, this function does nothing.

f.delete This function sends the WM_DELETE_WINDOW message to the selected window if the client application has requested it through the WM_PROTOCOLS window property. The application is supposed to respond to the message by removing the indicated window. If the window has not requested WM_DELETE_WINDOW messages, the keyboard bell will be rung indicating that the user should choose an alternative method.

f.deltastop
This function allows a user-defined function to be aborted if the pointer has been moved more than *MoveDelta* pixels. See the example definition given for **Function "move-or-raise"** at the beginning of the section.

f.destroy
This function instructs the X server to close the display connection of the client that created the selected window. This should only be used as a last resort for shutting down runaway clients.

f.downiconmgr
This function warps the pointer to the next row in the current icon manger, wrapping to the beginning of the next column if necessary.

f.exec *string*
This function passes the argument *string* to /bin/sh for execution. In multiscreen mode, if *string* starts a new X client without giving a display argument, the client will appear on the screen from which this function was invoked.

f.focus This function toggles the keyboard focus of the server to the selected window, changing the focus rule from pointer-driven if necessary. If the selected window already was focused, this function executes an **f.unfocus**.

f.forcemove
This function is like **f.move** except that it ignores the **DontMoveOff** variable.

f.forwiconmgr
This function warps the pointer to the next column in the current icon manager, wrapping to the beginning of the next row if necessary.

f.fullzoom
This function resizes the selected window to the full size of the display or else restores the original size if the window was already zoomed.

f.function *string*
This function executes the user-defined function whose name is specified by the argument *string*.

f.hbzoom
This function is a synonym for **f.bottomzoom**.

f.hideiconmgr
This function unmaps the current icon manager.

f.horizoom
This variable is similar to the **f.zoom** function except that the selected window is resized to the full width of the display.

f.htzoom
This function is a synonym for **f.topzoom**.

f.hzoom
This function is a synonym for **f.horizoom**.

f.iconify
This function iconifies or deiconifies the selected window or icon, respectively.

f.identify
This function displays a summary of the name and geometry of the selected window. Clicking the pointer or pressing a key in the window will dismiss it.

f.lefticonmgr
This function similar to **f.backiconmgr** except that wrapping does not change rows.

f.leftzoom
This variable is similar to the **f.bottomzoom** function but causes the selected window is only resized to the left half of the display.

f.lower This function lowers the selected window.

f.menu *string*
This function invokes the menu specified by the argument *string*. Cascaded menus may be built by nesting calls to **f.menu**.

f.move This function drags an outline of the selected window (or the window itself if the **OpaqueMove** variable is set) until the invoking pointer button is released. Double clicking within the number of milliseconds given by **ConstrainedMoveTime** warps the pointer to the center of the window and constrains the move to be either horizontal or vertical

depending on which grid line is crossed. To abort a move, press another button before releasing the first button.

f.nexticonmgr
This function warps the pointer to the next icon manager containing any windows on the current or any succeeding screen.

f.nop This function does nothing and is typically used with the **DefaultFunction** or **WindowFunction** variables or to introduce blank lines in menus.

f.previconmgr
This function warps the pointer to the previous icon manager containing any windows on the current or preceding screens.

f.quit This function causes *twm* to restore the window's borders and exit. If *twm* is the first client invoked from *xdm*, this will result in a server reset.

f.raise This function raises the selected window.

f.raiselower
This function raises the selected window to the top of the stacking order if it is occluded by any windows, otherwise the window will be lowered.

f.refresh
This function causes all windows to be refreshed.

f.resize This function displays an outline of the selected window. Crossing a border (or setting **AutoRelativeResize**) will cause the outline to begin to rubber band until the invoking button is released. To abort a resize, press another button before releasing the first button.

f.restart
This function kills and restarts *twm*.

f.righticonmgr
This function is similar to **f.nexticonmgr** except that wrapping does not change rows.

f.rightzoom
This variable is similar to the **f.bottomzoom** function except that the selected window is only resized to the right half of the display.

f.saveyourself
This function sends a WM_SAVEYOURSELF message to the selected window if it has requested the message in its WM_PROTOCOLS window property. Clients that accept this message are supposed to checkpoint all state associated with the window and update the WM_COMMAND property as specified in the ICCCM. If the selected window has not selected for this message, the keyboard bell will be rung.

f.showiconmgr
This function maps the current icon manager.

f.sorticonmgr
This function sorts the entries in the current icon manager alphabetically. See the variable **SortIconManager**.

f.title This function provides a centered, unselectable item in a menu definition. It should not be used in any other context.

f.topzoom
This variable is similar to the **f.bottomzoom** function except that the selected window is only resized to the top half of the display.

f.unfocus
This function resets the focus back to pointer-driven. This should be used when a

focused window is no longer desired.

f.upiconmgr
This function warps the pointer to the previous row in the current icon manager, wrapping to the last row in the same column if necessary.

f.vlzoom
This function is a synonym for **f.leftzoom**.

f.vrzoom
This function is a synonym for **f.rightzoom**.

f.warpring *string*
This function warps the pointer to the next or previous window (as indicated by the argument *string*, which may be "**next**" or "**prev**") specified in the **WindowRing** variable.

f.warpto *string*
This function warps the pointer to the window which has a name or class that matches *string*. If the window is iconified, it will be deiconified if the variable **WarpUnmapped** is set or else ignored.

f.warptoiconmgr *string*
This function warps the pointer to the icon manager entry associated with the window containing the pointer in the icon manager specified by the argument *string*. If *string* is empty (i.e. ""), the current icon manager is chosen.

f.warptoscreen *string*
This function warps the pointer to the screen specified by the argument *string*. *String* may be a number (e.g. "**0**" or "**1**"), the word "**next**" (indicating the current screen plus 1, skipping over any unmanaged screens), the word "**back**" (indicating the current screen minus 1, skipping over any unmanaged screens), or the word "**prev**" (indicating the last screen visited.

f.winrefresh
This function is similar to the **f.refresh** function except that only the selected window is refreshed.

f.zoom This function is similar to the **f.fullzoom** function, except that the only the height of the selected window is changed.

MENUS

Functions may be grouped and interactively selected using pop-up (when bound to a pointer button) or pull-down (when associated with a titlebutton) menus. Each menu specification contains the name of the menu as it will be referred to by **f.menu**, optional default foreground and background colors, the list of item names and the functions they should invoke, and optional foreground and background colors for individual items:

Menu "*menuname*" [("*deffore*":"*defback*")]
{
string1 [("*fore1*":"*backn*")] *function1*
string2 [("*fore2*":"*backn*")] *function2*
.
.
.
stringN [("*foreN*":"*backN*")] *functionN*
}

The *menuname* is case-sensitive. The optional *deffore* and *defback* arguments specify the foreground and background colors used on a color display to highlight menu entries. The *string* portion of each menu entry will be the text which will appear in the menu. The optional *fore* and *back* arguments specify the foreground and background colors of the menu entry when the pointer is not in the entry. These colors will only be used on a color display. The default is to use the colors specified by the **MenuForeground** and **MenuBackground** variables. The *function* portion of the menu entry is one of the functions, including any user-defined functions, or additional menus.

There is a special menu named **TwmWindows** which contains the names of all of the client and *twm*-supplied windows. Selecting an entry will cause the **WindowFunction** to be executed on that window. If **WindowFunction** hasn't been set, the window will be deiconified and raised.

ICONS

Twm supports several different ways of manipulating iconified windows. The common pixmap-and-text style may be laid out by hand or automatically arranged as described by the **IconRegion** variable. In addition, a terse grid of icon names, called an icon manager, provides a more efficient use of screen space as well as the ability to navigate among windows from the keyboard.

An icon manager is a window that contains names of selected or all windows currently on the display. In addition to the window name, a small button using the default iconify symbol will be displayed to the left of the name when the window is iconified. By default, clicking on an entry in the icon manager performs **f.iconify**. To change the actions taken in the icon manager, use the the **iconmgr** context when specifying button and keyboard bindings.

Moving the pointer into the icon manager also directs keyboard focus to the indicated window (setting the focus explicitly or else sending synthetic events **NoTitleFocus** is set). Using the **f.upiconmgr**, **f.downiconmgr** **f.lefticonmgr**, and **f.righticonmgr** functions, the input focus can be changed between windows directly from the keyboard.

BUGS

The resource manager should have been used instead of all of the window lists.

The **IconRegion** variable should take a list.

Double clicking very fast to get the constrained move function will sometimes cause the window to move, even though the pointer is not moved.

If **IconifyByUnmapping** is on and windows are listed in **IconManagerDontShow** but not in **DontIconifyByUnmapping**, they may be lost if they are iconified and no bindings to **f.menu "TwmWindows"** or **f.warpto** are setup.

FILES

$HOME/.twmrc.<screen number>
$HOME/.twmrc
/usr/lib/X11/twm/system.twmrc

ENVIRONMENT VARIABLES

DISPLAY
: This variable is used to determine which X server to use. It is also set during **f.exec** so that programs come up on the proper screen.

HOME
: This variable is used as the prefix for files that begin with a tilde and for locating the *twm* startup file.

SEE ALSO

X(1), Xserver(1), xdm(1), xrdb(1)

COPYRIGHT

statement of rights and permissions.

AUTHORS

Tom LaStrange, Solbourne Computer; Jim Fulton, MIT X Consortium; Steve Pitschke, Stardent Computer; Keith Packard, MIT X Consortium; Dave Payne, Apple Computer.

NOTE

twm is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xauth - X authority file utility

SYNOPSIS

xauth [-f *authfile*] [-vqib] [*command arg...*]

DESCRIPTION

The *xauth* program is used to edit and display the authorization information used in connecting to the X server. This program is usually to extract authorization records from one machine and merge them in on another (as is the case when using remote logins or to grant access to other users). Commands (described below) may be entered interactively, on the *xauth* command line, or in scripts. Note that this program does **not** contact the X server.

OPTIONS

The following options may be used with *xauth*. They may be given individually (e.g. *-q -i*) or may combined (e.g. *-qi*):

-f *authfile*
: This option specifies the name of the authority file to use. By default, *xauth* will use the file specified by the XAUTHORITY environment variable or *.Xauthority* in the user's home directory.

-q
: This option indicates that *xauth* should operate quietly and not print unsolicited status messages. This is the default if an *xauth* command is is given on the command line or if the standard output is not directed to a terminal.

-v
: This option indicates that *xauth* should operate verbosely and print status messages indicating the results of various operations (e.g. how many records have been read in or written out). This is the default if *xauth* is reading commands from its standard input and its standard output is directed to a terminal.

-i
: This option indicates that *xauth* should ignore any authority file locks. Normally, *xauth* will refuse to read or edit any authority files that have been locked by other programs (usually *xdm* or another *xauth*).

-b
: This option indicates that *xauth* should attempt to break any authority file locks before proceeding and should only be used to clean up stale locks.

COMMANDS

The following commands may be used to manipulate authority files:

add *displayname protocolname hexkey*
: An authorization entry for the indicated display using the given protocol and key data is added to the authorization file. The data is specified as an even-lengthed string of hexadecimal digits, each pair representing one octet. The first digit gives the most significant 4 bits of the octet and the second digit gives the least significant 4 bits. A protocol name consisting of just a single period is treated as an abbreviation for *MIT-MAGIC-COOKIE-1*.

[n]extract *filename displayname...*
: Authorization entries for each of the specified displays are written to the indicated file. If the *nextract* command is used, the entries are written in a numeric format suitable for non-binary transmission (such as secure electronic mail). The extracted entries can be read back in using the *merge* and *nmerge* commands. If the the filename consists of just a single dash, the entries will be written to the standard output.

[n]list [*displayname...*]
: Authorization entries for each of the specified displays (or all if no displays are named) are printed on the standard output. If the *nlist* command is used, entries will be shown in the numeric format used by the *nextract* command; otherwise, they are shown in a textual format. Key data is always displayed in the hexadecimal format given in the

description of the *add* command.

[n]merge [*filename...*]
Authorization entries are read from the specified files and are merged into the authorization database, superceding any matching existing entries. If the *nmerge* command is used, the numeric format given in the description of the *extract* command is used. If a filename consists of just a single dash, the standard input will be read if it hasn't been read before.

remove *displayname...*
Authorization entries matching the specified displays are removed from the authority file.

source *filename*
The specified file is treated as a script containing *xauth* commands to execute. Blank lines and lines beginning with a sharp sign (#) are ignored. A single dash may be used to indicate the standard input, if it hasn't already been read.

info Information describing the authorization file, whether or not any changes have been made, and from where *xauth* commands are being read is printed on the standard output.

exit If any modifications have been made, the authority file is written out (if allowed), and the program exits. An end of file is treated as an implicit *exit* command.

quit The program exits, ignoring any modifications. This may also be accomplished by pressing the interrupt character.

help [*string*]
A description of all commands that begin with the given string (or all commands if no string is given) is printed on the standard output.

? A short list of the valid commands is printed on the standard output.

DISPLAY NAMES

Display names for the *add*, *[n]extract*, *[n]list*, *[n]merge*, and *remove* commands use the same format as the DISPLAY environment variable and the common *-display* command line argument. Display-specific information (such as the screen number) is unnecessary and will be ignored. Same-machine connections (such as local-host sockets, shared memory, and the Internet Protocol hostname *localhost*) are referred to as *hostname*/unix:*displaynumber* so that local entries for different machines may be stored in one authority file.

EXAMPLE

The most common use for *xauth* is to extract the entry for the current display, copy it to another machine, and merge it into the user's authority file on the remote machine:

```
% xauth extract - $DISPLAY | rsh other xauth merge -
```

ENVIRONMENT

This *xauth* program uses the following environment variables:

XAUTHORITY
to get the name of the authority file to use if the *-f* option isn't used. If this variable is not set, *xauth* will use *.Xauthority* in the user's home directory.

HOME to get the user's home directory if XAUTHORITY isn't defined.

BUGS

Users that have unsecure networks should take care to use encrypted file transfer mechanisms to copy authorization entries between machines. Similarly, the *MIT-MAGIC-COOKIE-1* protocol is not very useful in unsecure environments. Sites that are interested in additional security may need to use encrypted authorization mechanisms such as Kerberos.

Spaces are currently not allowed in the protocol name. Quoting could be added for the truly perverse.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

xauth is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xbiff - mailbox flag for X

SYNOPSIS

xbiff [-*toolkitoption* ...] [-option ...]

DESCRIPTION

The *xbiff* program displays a little image of a mailbox. When there is no mail, the flag on the mailbox is down. When mail arrives, the flag goes up and the mailbox beeps. By default, pressing any mouse button in the image forces *xbiff* to remember the current size of the mail file as being the "empty" size and to lower the flag.

This program is nothing more than a wrapper around the Athena *Mailbox* widget.

OPTIONS

Xbiff accepts all of the standard X Toolkit command line options along with the additional options listed below:

-help This option indicates that a brief summary of the allowed options should be printed on the standard error.

-update *seconds*
This option specifies the frequency in seconds at which *xbiff* should update its display. If the mailbox is obscured and then exposed, it will be updated immediately. The default is 60 seconds.

-file *filename*
This option specifies the name of the file which should be monitored. By default, it watches /usr/spool/mail/*username*, where *username* is your login name.

-volume *percentage*
This option specifies how loud the bell should be rung when new mail comes in.

-shape This option indicates that the mailbox window should be shaped if masks for the empty or full images are given.

The following standard X Toolkit command line arguments are commonly used with *xbiff:*

-display *display*
This option specifies the X server to contact.

-geometry *geometry*
This option specifies the prefered size and position of the mailbox window. The mailbox is 48 pixels wide and 48 pixels high and will be centered in the window.

-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 the foreground of the window. The default is "black."

-rv This option indicates that reverse video should be simulated by swapping the foreground and background colors.

-xrm *resourcestring*

This option specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

This program uses the *Mailbox* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

checkCommand (class **CheckCommand**)
Specifies a shell command to be executed to check for new mail rather than examining the size of **file**. The specified string value is used as the argument to a *system*(3) call and may therefore contain i/o redirection. An exit status of 0 indicates that new mail is waiting, 1 indicates that there has been no change in size, and 2 indicates that the mail has been cleared.

file (class **File**)
Specifies the name of the file to monitor. The default is to watch /usr/spool/mail/*username*, where *username* is your login name.

onceOnly (class **Boolean**)
Specifies that the bell is only rung the first time new mail is found and is not rung again until at least one interval has passed with no mail waiting. The window will continue to indicate the presence of new mail until it has been retrieved.

width (class **Width**)
Specifies the width of the mailbox.

height (class **Height**)
Specifies the height of the mailbox.

update (class **Interval**)
Specifies the frequency in seconds at which the mail should be checked.

volume (class **Volume**)
Specifies how load the bell should be rung. The default is 33 percent.

foreground (class **Foreground**)
Specifies the color for the foreground. The default is "black" since the core default for background is "white."

reverseVideo (class **ReverseVideo**)
Specifies that the foreground and background should be reversed.

flip (class **Flip**)
Specifies whether or not the image that is shown when mail has arrived should be inverted. The default is "true."

fullPixmap (class **Pixmap**)
Specifies a bitmap to be shown when new mail has arrived.

emptyPixmap (class **Pixmap**)
Specifies a bitmap to be shown when no new mail is present.

shapeWindow (class **ShapeWindow**)
Specifies whether or not the mailbox window should be shaped to the given fullPixmap-Mask and emptyPixmapMask.

fullPixmapMask (class **PixmapMask**)
Specifies a mask for the bitmap to be shown when new mail has arrived.

emptyPixmapMask (class **PixmapMask**)
Specifies a mask for the bitmap to be shown when no new mail is present.

ACTIONS

The *Mailbox* widget provides the following actions for use in event translations:

check() This action causes the widget to check for new mail and display the flag appropriately.

unset() This action causes the widget to lower the flag until new mail comes in.

set() This action causes the widget to raise the flag until the user resets it.

The default translation is

```
<ButtonPress>: unset()
```

ENVIRONMENT

DISPLAY

to get the default host and display number.

XENVIRONMENT

to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1), xrdb(1), stat(2)

BUGS

The mailbox bitmaps are ugly.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium
Additional hacks by Ralph Swick, DEC/MIT Project Athena

NOTE

xbiff is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xcalc - scientific calculator for X

SYNOPSIS

xcalc [-stipple] [-rpn] [-*toolkitoption*...]

DESCRIPTION

xcalc is a scientific calculator desktop accessory that can emulate a TI-30 or an HP-10C.

OPTIONS

xcalc accepts all of the standard toolkit command line options along with two additional options:

-stipple
This option indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays improves the appearance.

-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.

OPERATION

Pointer Usage: Operations may be performed with pointer button 1, or in some cases, with the keyboard. Many common calculator operations have keyboard accelerators. To quit, press pointer button 3 on the AC key of the TI calculator, or the ON key of the HP calculator.

Calculator Key Usage (TI mode): The numbered 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 entire number in the calculator display can be selected, in order to paste the result of a calculation into text.

The action procedures associated with each function are given below. These are useful if you are interested in defining a custom calculator. The action used for all digit keys is **digit(***n***)**, where *n* is the corresponding digit, 0..9.

1/x Replaces the number in the display with its reciprocal. The corresponding action procedure is **reciprocal()**.

x^2 Squares the number in the display. The corresponding action procedure is **square()**.

SQRT Takes the square root of the number in the display. The corresponding action procedure is **squareRoot()**.

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 make a mistake. Pressing it twice clears the state, also. The corresponding action procedure for TI mode is **clear()**.

AC Clears the display, the state, and the memory. Pressing it with the third pointer button turns off the calculator, in that it exits the program. The action procedure to clear the state is **off()**; to quit, **quit()**.

INV Invert function. See the individual function keys for details. The corresponding action procedure is **inverse()**.

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. The corresponding action procedure is **sine()**.

cos Computes the cosine, or arccosine when inverted. The corresponding action procedure is **cosine()**.

tan Computes the tangent, or arctangent when inverted. The corresponding action procedure is **tangent()**.

DRG Changes the DRG mode, as indicated by 'DEG', 'RAD', or 'GRAD' at the bottom of of the calculator "liquid crystal" display. When in 'DEG' mode, numbers in the display are taken as being degrees. In 'RAD' mode, numbers are in radians, and in 'GRAD' mode, numbers are in grads. When inverted, the DRG key has a feature of converting degrees to radians to grads and vice-versa. Example: put the calculator into 'DEG' mode, and enter "45 INV DRG". The display should now show something along the lines of ".785398", which is 45 degrees converted to radians. The corresponding action procedure is **degree()**.

e The constant 'e'. (2.7182818...). The corresponding action procedure is **e()**.

EE Used for entering exponential numbers. For example, to get "-2.3E-4" you'd enter "2 . 3 +/- EE 4 +/-". The corresponding action procedure is **scientific()**.

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, entering "3 INV log" should result in "1000". The corresponding action procedure is **logarithm()**.

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, entering "e ln" should result in "1". The corresponding action procedure is **naturalLog()**.

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 2^3. For a further example, "(1+2+3) y^x (1+2) =" equals "6 y^x 3" which equals "216". The corresponding action procedure is **power()**.

PI The constant 'pi'. (3.1415927....) The corresponding action procedure is **pi()**.

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. The corresponding action procedure is **factorial()**.

(Left parenthesis. The corresponding action procedure for TI calculators is **leftParen()**.

) Right parenthesis. The corresponding action procedure for TI calculators is **rightParen()**.

/ Division. The corresponding action procedure is **divide()**.

***** Multiplication. The corresponding action procedure is **multiply()**.

- Subtraction. The corresponding action procedure is **subtract()**.

+ Addition. The corresponding action procedure is **add()**.

= Perform calculation. The TI-specific action procedure is **equal()**.

STO Copies the number in the display to the memory location. The corresponding action procedure is **store()**.

RCL Copies the number from the memory location to the display. The corresponding action procedure is **recall()**.

SUM Adds the number in the display to the number in the memory location. The corresponding action procedure is **sum()**.

EXC Swaps the number in the display with the number in the memory location. The corresponding action procedure for the TI calculator is **exchange()**.

+/- Negate; change sign. The corresponding action procedure is **negate()**.

. Decimal point. The action procedure is **decimal()**.

Calculator 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 TI mode. The differences are detailed below. The action procedure for the ENTR key is **enter()**.

<- This is a backspace key that can be used if you make a mistake while entering a number. It will erase digits from the display. (See BUGS). Inverse backspace will clear the X register. The corresponding action procedure is **back()**.

ON Clears the display, the state, and the memory. Pressing it with the third pointer button turns off the calculator, in that it exits the program. To clear state, the action procedure is **off**; to quit, **quit()**.

INV Inverts the meaning of the function keys. This would be the *f* key on an HP calculator, but *xcalc* does not display multiple legends on each key. See the individual function keys for details.

10^x 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. The corresponding action procedure is **tenpower()**.

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. The action procedure is **epower()**.

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 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, the X and Y registers. The corresponding action procedure is **XexchangeY()**.

R v Rolls the stack downward. When inverted, it rolls the stack upward. The corresponding action procedure is **roll()**.

blank These keys were used for programming functions on the HP-10C. Their functionality has not been duplicated in *xcalc*.

Finally, there are two additional action procedures: **bell()**, which rings the bell; and **selection()**, which performs a cut on the entire number in the calculator's "liquid crystal" display.

ACCELERATORS

Accelerators are shortcuts for entering commands. *xcalc* provides some sample keyboard accelerators; also users can customize accelerators. The numeric keypad accelerators provided by *xcalc* should be intuitively correct. The accelerators defined by *xcalc* on the main keyboard are given below:

TI Key	HP Key	Keyboard Accelerator	TI Function	HP Function
SQRT	SQRT	r	squareRoot()	squareRoot()
AC	ON	space	clear()	clear()
AC	<-	Delete	clear()	back()
AC	<-	Backspace	clear()	back()
AC	<-	Control-H	clear()	back()
AC		Clear	clear()	

AC	ON	q	quit()	quit()
AC	ON	Control-C	quit()	quit()
INV	i	i	inverse()	inverse()
sin	s	s	sine()	sine()
cos	c	c	cosine()	cosine()
tan	t	t	tangent()	tangent()
DRG	DRG	d	degree()	degree()
e		e	e()	
ln	ln	l	naturalLog()	naturalLog()
y^x	y^x	^	power()	power()
PI	PI	p	pi()	pi()
x!	x!	!	factorial()	factorial()
(		(	leftParen()	
)		)	rightParen()	
/	/	/	divide()	divide()
*	*	*	multiply()	multiply()
-	-	-	subtract()	subtract()
+	+	+	add()	add()
=		=	equal()	
0..9	0..9	0..9	digit()	digit()
.	.	.	decimal()	decimal()
+/-	CHS	n	negate()	negate()
	x:y	x		XexchangeY()
	ENTR	Return		enter()
	ENTR	Linefeed		enter()

CUSTOMIZATION

The application class name is XCalc.

xcalc has an enormous application defaults file which specifies the position, label, and function of each key on the calculator. It also gives translations to serve as keyboard accelerators. Because these resources are not specified in the source code, you can create a customized calculator by writing a private application defaults file, using the Athena Command and Form widget resources to specify the size and position of buttons, the label for each button, and the function of each button.

The foreground and background colors of each calculator key can be individually specified. For the TI calculator, a classical color resource specification might be:

XCalc.ti.Command.background: gray50
XCalc.ti.Command.foreground: white

For each of buttons 20, 25, 30, 35, and 40, specify:
XCalc.ti.button20.background: black
XCalc.ti.button20.foreground: white

For each of buttons 22, 23, 24, 27, 28, 29, 32, 33, 34, 37, 38, and 39:
XCalc.ti.button22.background: white
XCalc.ti.button22.foreground: black

WIDGET HIERARCHY

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xcalc*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XCalc xcalc
        Form ti or rpn   (the name depends on the mode)
                Form bevel
                        Form screen
                                Label M
                                Toggle LCD
                                Label INV
                                Label DEG
                                Label RAD
                                Label GRAD
                                Label P
                Command button1
                Command button2
                Command button3
and so on, ...
                Command button38
                Command button39
                Command button40
```

APPLICATION RESOURCES

rpn (Class **Rpn**)

Specifies that the rpn mode should be used. The default is TI mode.

stipple (Class **Stipple**)

Indicates that the background should be stippled. The default is "on" for monochrome displays, and "off" for color displays.

cursor (Class **Cursor**)

The name of the symbol used to represent the pointer. The default is "hand2".

SEE ALSO

X(1), xrdb(1), the Athena Widget Set

BUGS

HP mode: A bug report claims that the sequence of keys 5, ENTER, <- should clear the display, but it doesn't.

COPYRIGHT

AUTHORS

John Bradley, University of Pennsylvania
Mark Rosenstein, MIT Project Athena
Donna Converse, MIT X Consortium

NOTE

xcalc is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xclipboard - X clipboard client

SYNOPSIS

xclipboard [*-toolkitoption* ...] [-w] [-nw]

DESCRIPTION

The *xclipboard* program is used to collect and display text selections that are sent to the CLIPBOARD by other clients. It is typically used to save CLIPBOARD selections for later use. It stores each CLIPBOARD selection as a separate string, each of which can be selected. Each time CLIPBOARD is asserted by another application, *xclipboard* transfers the contents of that selection to a new buffer and displays it in the text window. Buffers are never automatically deleted, so you'll want to use the delete button to get rid of useless items.

Since *xclipboard* uses a Text Widget to display the contents of the clipboard, text sent to the CLIPBOARD may be re-selected for use in other applications. *xclipboard* also responds to requests for the CLIPBOARD selection from other clients by sending the entire contents of the currently displayed buffer.

An *xclipboard* window has the following buttons across the top:

quit When this button is pressed, *xclipboard* exits.

delete When this button is pressed, the current buffer is deleted and the next one displayed.

new Creates a new buffer with no contents. Useful in constructing a new CLIPBOARD selection by hand.

next Displays the next buffer in the list.

previous Displays the previous buffer.

OPTIONS

The *xclipboard* program accepts all of the standard X Toolkit command line options as well as the following:

-w This option indicates that lines of text that are too long to be displayed on one line in the clipboard should wrap around to the following lines.

-nw This option indicates that long lines of text should not wrap around. This is the default behavior.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xclipboard*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XClipboard xclipboard
        Form form
                Command quit
                Command delete
                Command new
                Command next
                Command prev
                Text text
```

SENDING/RETRIEVING CLIPBOARD CONTENTS

Text is copied to the clipboard whenever a client asserts ownership of the **CLIPBOARD** selection. Text is copied from the clipboard whenever a client requests the contents of the **CLIPBOARD** selection. Examples of event bindings that a user may wish to include in a resource configuration file to use the clipboard are:

```
*VT100.Translations: #override \
        <Btn3Up>:               select-end(CLIPBOARD) \n\
        <Btn2Up>:               insert-selection(PRIMARY,CLIPBOARD) \n\
        <Btn2Down>:             ignore ()
```

SEE ALSO

X(1), xcutsel(1), xterm(1), individual client documentation for how to make a selection and send it to the CLIPBOARD.

ENVIRONMENT

DISPLAY

to get the default host and display number.

XENVIRONMENT

to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

/usr/lib/X11/app-defaults/XClipboard - specifies required resources

COPYRIGHT

AUTHOR

Ralph R. Swick, DEC/MIT Project Athena
Chris D. Peterson, MIT X Consortium
Keith Packard, MIT X Consortium

NOTE

xclipboard is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xclock - analog / digital clock for X

SYNOPSIS

xclock [-*toolkitoption* ...] [-help] [-analog] [-digital] [-chime] [-hd *color*] [-hl *color*] [-update *seconds*] [-padding *number*]

DESCRIPTION

The *xclock* program displays the time in analog or digital form. The time is continuously updated at a frequency which may be specified by the user. This program is nothing more than a wrapper around the Athena Clock widget.

OPTIONS

Xclock accepts all of the standard X Toolkit command line options along with the additional options listed below:

-help This option indicates that a brief summary of the allowed options should be printed on the standard error.

-analog
This option indicates that a conventional 12 hour clock face with tick marks and hands should be used. This is the default.

-digital This option indicates that a 24 hour digital clock should be used.

-chime This option indicates that the clock should chime once on the half hour and twice on the hour.

-hd *color*
This option specifies the color of the hands on an analog clock. The default is *black*.

-hl *color*
This option specifies the color of the edges of the hands on an analog clock, and is only useful on color displays. The default is *black*.

-update *seconds*
This option specifies the frequency in seconds at which *xclock* should update its display. If the clock is obscured and then exposed, it will be updated immediately. A value of less than 30 seconds will enable a second hand on an analog clock. The default is 60 seconds.

-padding *number*
This option specifies the width in pixels of the padding between the window border and clock text or picture. The default is 10 on a digital clock and 8 on an analog clock.

X DEFAULTS

This program uses the *Athena Clock* widget. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

height (class **Height**)
Specifies the height of the clock. The default for analog clocks is 164 pixels; the default for digital clocks is whatever is needed to hold the clock when displayed in the chosen font.

update (class **Interval**)
Specifies the frequency in seconds at which the time should be redisplayed.

foreground (class **Foreground**)

Specifies the color for the tic marks. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is *black*.

hands (class **Foreground**)
Specifies the color of the insides of the clock's hands. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is *black*.

highlight (class **Foreground**)
Specifies the color used to highlight the clock's hands. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is *black*.

analog (class **Boolean**)
Specifies whether or not an analog clock should be used instead of a digital one. The default is True.

chime (class **Boolean**)
Specifies whether or not a bell should be rung on the hour and half hour.

padding (class **Margin**)
Specifies the amount of internal padding in pixels to be used. The default is 8.

font (class **Font**)
Specifies the font to be used for the digital clock. Note that variable width fonts currently will not always display correctly.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xclock*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XClock  xclock
        Clock  clock
```

ENVIRONMENT

DISPLAY
to get the default host and display number.

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

FILES

/usr/lib/X11/app-defaults/XClock - specifies required resources

SEE ALSO

X(1), xrdb(1), time(3C), Athena Clock widget

BUGS

Xclock believes the system clock.

When in digital mode, the string should be centered automatically.

COPYRIGHT

AUTHORS

Tony Della Fera (MIT-Athena, DEC)

Dave Mankins (MIT-Athena, BBN)
Ed Moy (UC Berkeley)

NOTE

xclock is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xcutsel - interchange between cut buffer and selection

SYNOPSIS

xcutsel [*-toolkitoption* ...] [-selection *selection*] [-cutbuffer *number*]

DESCRIPTION

The *xcutsel* program is used to copy the current selection into a cut buffer and to make a selection that contains the current contents of the cut buffer. It acts as a bridge between applications that don't support selections and those that do.

By default, *xcutsel* will use the selection named PRIMARY and the cut buffer CUT_BUFFER0. Either or both of these can be overridden by command line arguments or by resources.

An *xcutsel* window has the following buttons:

quit When this button is pressed, *xcutsel* exits. Any selections held by *xcutsel* are automatically released.

copy PRIMARY to 0
When this button is pressed, *xcutsel* copies the current selection into the cut buffer.

copy 0 to PRIMARY
When this button is pressed, *xcutsel* converts the current contents of the cut buffer into the selection.

The button labels reflect the selection and cutbuffer selected by command line options or through the resource database.

When the "copy 0 to PRIMARY" button is activated, the button will remain inverted as long as *xcutsel* remains the owner of the selection. This serves to remind you which client owns the current selection. Note that the value of the selection remains constant; if the cutbuffer is changed, you must again activate the copy button to retrieve the new value when desired.

OPTIONS

Xcutsel accepts all of the standard X Toolkit command line options as well as the following:

–selection *name*
This option specifies the name of the selection to use. The default is PRIMARY. The only supported abbreviations for this option are "-select", "-sel" and "-s", as the standard toolkit option "-selectionTimeout" has a similar name.

–cutbuffer *number*
This option specifies the cut buffer to use. The default is cut buffer 0.

X DEFAULTS

This program accepts all of the standard X Toolkit resource names and classes as well as:

selection (class **Selection**)
This resource specifies the name of the selection to use. The default is PRIMARY.

cutBuffer (class **CutBuffer**)
This resource specifies the number of the cut buffer to use. The default is 0.

WIDGET NAMES

The following instance names may be used when user configuration of the labels in them is desired:

sel-cut (class **Command**)
This is the "copy SELECTION to BUFFER" button.

cut-sel (class **Command**)
This is the "copy BUFFER to SELECTION" button.

quit (class **Command**)

This is the "quit" button.

SEE ALSO

X(1), xclipboard(1), xterm(1), text widget documentation, individual client documentation for how to make a selection.

BUGS

There is no way to change the name of the selection or the number of the cut buffer while the program is running.

AUTHOR

Ralph R. Swick, DEC/MIT Project Athena

NAME

xdm - X Display Manager

SYNOPSIS

xdm [-config *configuration_file*] [-daemon] [-debug *debug_level*] [-error *error_log_file*] [-nodaemon] [-resources *resource_file*] [-server *server_entry*] [-session *session_program*] [-xrm *resource_specification*]

DESCRIPTION

Xdm manages a collection of X displays, both local and possibly remote — the emergence of X terminals guided the design of several parts of this system, along with the development of the X Consortium standard XDMCP, the *X Display Manager Control Protocol*). It is designed to provide services similar to that provided by init, getty and login on character terminals: prompting for login/password, authenticating the user and running a "session".

A "session" is defined by the lifetime of a particular process; in the traditional character-based terminal world, it is the user's login shell process. In the *xdm* context, it is an arbitrary session manager. This is because in a windowing environment, a user's login shell process would not necessarily have any terminal-like interface with which to connect.

Until real session managers become widely available, the typical *xdm* substitute would be either a window manager with an exit option, or a terminal emulator running a shell - with the condition that the lifetime of the terminal emulator is the lifetime of the shell process that it is running - thus degenerating the X session to an emulation of the character-based terminal session.

When the session is terminated, *xdm* resets the X server and (optionally) restarts the whole process.

Because *xdm* provides the first interface that users will see, it is designed to be simple to use and easy to customize to the needs of a particular site. *Xdm* has many options, most of which have reasonable defaults. Browse through the various sections, picking and choosing the things you want to change. Pay particular attention to the **Xsession** section, which will describe how to set up the style of session desired.

OPTIONS

First, note that all of these options, except **-config**, specify values which can also be specified in the configuration file as resources.

-config *configuration_file*
: Specifies a resource file which specifies the remaining configuration parameters. If no file is specified and the file */usr/lib/X11/xdm/xdm-config* exists, *xdm* will use it.

-daemon
: Specifies "true" as the value for the **DisplayManager.daemonMode** resource. This makes *xdm* close all file descriptors, disassociate the controlling terminal and put itself in the background when it first starts up (just like the host of other daemons). It is the default behavior.

-debug *debug_level*
: Specifies the numeric value for the **DisplayManager.debugLevel** resource. A non-zero value causes *xdm* to print piles of debugging statements to the terminal; it also disables the **DisplayManager.daemonMode** resource, forcing *xdm* to run synchronously. To interpret these debugging messages, a copy of the source code for xdm is almost a necessity. No attempt has been made to rationalize or standardize the output.

-error *error_log_file*
: Specifies the value for the **DisplayManager.errorLogFile** resource. This file contains errors from *xdm* as well as anything written to stderr by the various scripts and programs run during the progress of the session.

-nodaemon

Specifies "false" as the value for the **DisplayManager.daemonMode** resource.

-resources *resource_file*
Specifies the value for the **DisplayManager*resources** resource. This file is loaded using *xrdb (1)* to specify configuration parameters for the authentication widget.

-server *server_entry*
Specifies the value for the **DisplayManager.servers** resource. See the section below which describes this resource in depth.

-udpPort *port_number*
Specifies the value for the **DisplayManager.requestPort** resource. This sets the port-number which XDM will monitor for XDMCP requests. As XDMCP uses the registered well-known udp port 177, this resource should probably not be changed except for debugging.

-session *session_program*
Specifies the value for the **DisplayManager*session** resource. This indicates the program to run when the user has logged in as the session.

-xrm *resource_specification*
This allows an arbitrary resource to be specified, just as most toolkit applications.

RESOURCES

At many stages the actions of *xdm* can be controlled through the use of the configuration file, which is in the familiar X resource format. See Jim Fulton's article on resource files (*doc/tutorials/resources.txt*) for a description of the format. Some resources modify the behavior of *xdm* on all displays, while others modify its behavior on one single display. Where actions relate to a specific display, the display name is inserted into the resource name between "DisplayManager" and the final resource name segment. For example, **DisplayManager.expo_0.startup** is the name of the resource which defines the startup shell file on the "expo:0" display. Because the resource manager uses colons to separate the name of the resource from its value and dots to separate resource name parts, *xdm* substitutes underscores for the dots and colons when generating the resource name.

DisplayManager.servers
This resource either specifies a file name full of server entries, one per line (if the value starts with a slash), or a single server entry. Each entry indicates a displays which should constantly be managed and which is not using XDMCP. Each entry consists of at least three parts: a display name, a display class, a display type, and (for local servers) a command line to start the server. A typical entry for local display number 0 would be:

```
:0 Digital-QV local /usr/bin/X11/X :0
```

The display types are:

local	a local display, i.e. one which has a server program to run
foreign	a remote display, i.e. one which has no server program to run

The display name must be something that can be passed in the **-display** option to any X program. This string is used in the display-specific resources to specify the particular display, so be careful to match the names (e.g. use ":0 local /usr/bin/X11/X :0" instead of "localhost:0 local /usr/bin/X11/X :0" if your other resources are specified as "DisplayManager._0.session"). The display class portion is also used in the display-specific resources, as the class portion of the resource. This is useful if you have a large collection of similar displays (like a corral of X terminals) and would like to set resources for groups of them. When using XDMCP, the display is required to specify the display

class, so perhaps your X terminal documentation describes a reasonably standard display class string for your device.

DisplayManager.requestPort
This indicates the UDP port number which *xdm* uses to listen for incoming XDMCP requests. Unless you need to debug the system, leave this with it's default value of 177.

DisplayManager.errorLogFile
Error output is normally directed at the system console. To redirect it simply set this resource to any file name. A method to send these messages to syslog should be developed for systems which support it; however the wide variety of "standard" interfaces precludes any system-independent implementation. This file also contains any output directed to stderr by *Xstartup, Xsession and Xreset*, so it will contain descriptions of problems in those scripts as well.

DisplayManager.debugLevel
A non-zero value specified for this integer resource will enable reams of debugging information to be printed. It also disables daemon mode which would redirect the information into the bit-bucket. Specifying a non-zero debug level also allows non-root users to run *xdm* which would normally not be useful.

DisplayManager.daemonMode
Normally, *xdm* attempts to make itself into an unassociated daemon process. This is accomplished by forking and leaving the parent process to exit, then closing file descriptors and mangling the controlling terminal. When attempting to debug *xdm* , this is quite bothersome. Setting this resource to "false" will disable this feature.

DisplayManager.pidFile
The filename specified will be created to contain an ascii representation of the process-id of the main **xdm** process. This is quite useful when reinitializing the system. *Xdm* also uses file locking to attempt to eliminate multiple daemons running on the same machine, which would cause quite a bit of havoc.

DisplayManager.lockPidFile
This is the resource which controls whether *xdm* uses file locking to keep multiple xdms from running amok. On SYSV, this uses the lockf library call, while on BSD it uses flock. The default value is "true".

DisplayManager.remoteAuthDir
This is a directory name which *xdm* uses to temporarily store authorization files for displays using XDMCP. The default value is /usr/lib/X11/xdm.

DisplayManager.autoRescan"
This boolean controls whether *xdm* rescans the configuration file and servers file after a session terminates and the files have changed. By default it is "true". You can force *xdm* to reread these files by sending a SIGHUP to the main process.

DisplayManager.removeDomainname
When computing the display name for XDMCP clients, the resolver will typically create a fully qualified host name for the terminal. As this is sometimes confusing, *xdm* will remove the domain name portion of the host name if it is the same as the domain name for the local host when this variable is set. By default the value is "true".

DisplayManager.keyFile
XDM-AUTHENTICATION-1 style XDMCP authentication requires that a private key be shared between *xdm* and the terminal. This resource specifies the file containing those values. Each entry in the file consists of a display name and the shared key. By default, *xdm* does not include support for XDM-AUTHENTICATION-1 as it requires DES which is not generally distributable.

DisplayManager.DISPLAY.resources
This resource specifies the name of the file to be loaded by *xrdb (1)* as the resource database onto the root window of screen 0 of the display. This resource data base is loaded just before the authentication procedure is started, so it can control the appearance of the "login" window. See the section below on the authentication widget which describes the various resources which are appropriate to place in this file. There is no default value for this resource, but the conventional name is **/usr/lib/X11/xdm/Xresources**.

DisplayManager.DISPLAY.xrdb
Specifies the program used to load the resources. By default, *xdm* uses */usr/bin/X11/xrdb*.

DisplayManager.DISPLAY.cpp
This specifies the name of the C preprocessor which is used by xrdb.

DisplayManager.DISPLAY.startup
This specifies a program which is run (as root) after the authentication process succeeds. By default, no program is run. The conventional name for a file used here is *Xstartup*. See the **Xstartup** section below.

DisplayManager.DISPLAY.session
This specifies the session to be executed (not running as root). By default, */usr/bin/X11/xterm* is run. The conventional name is *Xsession*. See the **Xsession** session below.

DisplayManager.DISPLAY.reset
This specifies a program which is run (as root) after the session terminates. Again, by default no program is run. The conventional name is *Xreset*. See the **Xreset** section further on in this document.

DisplayManager.DISPLAY.openDelay

DisplayManager.DISPLAY.openRepeat

DisplayManager.DISPLAY.openTimeout

DisplayManager.DISPLAY.startAttempts
These numeric resources control the behavior of *xdm* when attempting to open intransigent servers. **openDelay** is the length of the pause (in seconds) between successive attempts, **openRepeat** is the number of attempts to make, **openTimeout** is the amount of time to wait while actually attempting the open (i.e. the maximum time spent in the *connect (2)* syscall) and **startAttempts** is the number of times this entire process is done before giving up on the server. After **openRepeat** attempts have been made, or if **openTimeout** seconds elapse in any particular attempt, *xdm* terminates and restarts the server, attempting to connect again, this process is repeated **startAttempts** time, at which point the display is declared dead and disabled. Although this behavior may seem arbitrary, it has been empirically developed and works quite well on most systems. The default values are 5 for **openDelay**, 5 for **openRepeat**, 30 for **openTimeout** and 4 for **startAttempts**.

DisplayManager.DISPLAY.pingInterval

DisplayManager.DISPLAY.pingTimeout
To discover when remote displays disappear, *xdm* occasionally "pings" them, using an X connection and sending XSync requests. **pingInterval** specifies the time (in minutes) between each ping attempt, **pingTimeout** specifies the maximum amount of time (in minutes) to wait for the terminal to respond to the request. If the terminal does not respond, the session is declared dead and terminated. By default, both are set to 5 minutes. *xdm* will not ping local displays. Although it would seem harmless, it is unpleasant when the workstation session is terminated as a result of the server hanging

for NFS service and not responding to the ping.

DisplayManager.DISPLAY.terminateServer
This boolean resource specifies whether the X server should be terminated when a session terminates (instead of resetting it). This option can be used when the server tends to grow without bound over time in order to limit the amount of time the server is run. The default value is "FALSE".

DisplayManager.DISPLAY.userPath
Xdm sets the PATH environment variable for the session to this value. It should be a colon separated list of directories, see *sh(1)* for a full description. The default value can be specified in the X system configuration file with DefUserPath, frequently it is set to ":/bin:/usr/bin:/usr/bin/X11:/usr/ucb".

DisplayManager.DISPLAY.systemPath
Xdm sets the PATH environment variable for the startup and reset scripts to the value of this resource. The default for this resource is specified with the DefaultSystemPath entry in the system configuration file, but it is frequently "/etc:/bin:/usr/bin:/usr/bin/X11:/usr/ucb". Note the conspicuous absence of "." from this entry. This is a good practise to follow for root; it avoids many common trojan horse system penetration schemes.

DisplayManager.DISPLAY.systemShell
Xdm sets the SHELL environment variable for the startup and reset scripts to the value of this resource. By default, it is "/bin/sh".

DisplayManager.DISPLAY.failsafeClient
If the default session fails to execute, *xdm* will fall back to this program. This program is executed with no arguments, but executes using the same environment variables as the session would have had (see the section "Xsession" below). By default, */usr/bin/X11/xterm* is used.

DisplayManager.DISPLAY.grabServer

DisplayManager.DISPLAY.grabTimeout
To eliminate obvious security shortcomings in the X protocol, *xdm* grabs the server and keyboard while reading the name/password. The **grabServer** resource specifies if the server should be held for the duration of the name/password reading, when FALSE, the server is ungrabbed after the keyboard grab succeeds, otherwise the server is grabbed until just before the session begins. The **grabTimeout** resource specifies the maximum time *xdm* will wait for the grab to succeed. The grab may fail if some other client has the server grabbed, or possibly if the network latencies are very high. This resource has a default value of 3 seconds; you should be cautious when raising it as a user can be spoofed by a look-alike window on the display. If the grab fails, *xdm* kills and restarts the server (if possible) and session.

DisplayManager.DISPLAY.authorize

DisplayManager.DISPLAY.authName
authorize is a boolean resource which controls whether *xdm* generates and uses authorization for the server connections. If authorization is used, **authName** specifies the type to use. Currently, xdm supports only MIT-MAGIC-COOKIE-1 authorization, XDM-AUTHORIZATION-1 could be supported as well, but DES is not generally distributable. XDMCP connections specify which authorization types are supported dynamically, so **authName** is ignored in this case. When **authorize** is set for a display and authorization is not available, the user is informed by having a different message displayed in the login widget. By default, **authorize** is "true"; **authName** is "MIT-MAGIC-COOKIE-1".

DisplayManager.DISPLAY.authFile"

This file is used to communicate the authorization data from **xdm** to the server, using the *-auth* server command line option. It should be kept in a directory which is not world-writable as it could easily be removed, disabling the authorization mechanism in the server.

DisplayManager.DISPLAY.resetForAuth

The original implementation of authorization in the sample server reread the authorization file at server reset time, instead of when checking the initial connection. As *xdm* generates the authorization information just before connecting to the display, an old server would not get up-to-date authorization information. This resource causes *xdm* to send SIGHUP to the server after setting up the file, causing an additional server reset to occur, during which time the new authorization information will be read

DisplayManager.DISPLAY.userAuthDir

When *xdm* is unable to write to the usual user authorization file ($HOME/.Xauthority), it creates a unique file name in this directory and points the environment variable XAUTHORITY at the created file. By default it uses "/tmp".

CONTROLLING THE SERVER

Xdm controls local servers using POSIX signals. SIGHUP is expected to reset the server, closing all client connections and performing other clean up duties. SIGTERM is expected to terminate the server. If these signals do not perform the expected actions, *xdm* will not perform properly.

To control remote servers not using XDMCP, *xdm* searches the window hierarchy on the display and uses the protocol request KillClient in an attempt to clean up the terminal for the next session. This may not actually kill all of the clients, as only those which have created windows will be noticed. XDMCP provides a more sure mechanism; when xdm closes it's initial connection, the session is over and the terminal is required to close all other connections.

CONTROLLING XDM

Xdm responds to two signals: SIGHUP and SIGTERM. When sent a SIGHUP, *xdm* rereads the configuration file and the lfile specified by the **DisplayManager.servers** resource and notices if entries have been added or removed. If a new entry has been added, *xdm* starts a session on the associated display. Entries which have been removed are disabled immediately, meaning that any session in progress will be terminated without notice, and no new session will be started.

When sent a SIGTERM, *xdm* terminates all sessions in progress and exits. This can be used when shutting down the system.

Xdm attempts to mark the various sub-processes for ps(1) by editing the command line argument list in place. Because xdm can't allocate additional space for this task, it is useful to start xdm with a reasonably long command line (15 to 20 characters should be enough). Each process which is servicing a display is marked "-<Display-Name>".

AUTHENTICATION WIDGET

The authentication widget is an application which reads a name/password pair from the keyboard. As this is a toolkit client, nearly every imaginable parameter can be controlled with a resource. Resources for this widget should be put into the file named by **DisplayManager.DISPLAY.resources**. All of these have reasonable default values, so it is not necessary to specify any of them.

xlogin.Login.width, xlogin.Login.height, xlogin.Login.x, xlogin.Login.y

The geometry of the login widget is normally computed automatically. If you wish to position it elsewhere, specify each of these resources.

xlogin.Login.foreground

The color used to display the typed-in user name.

xlogin.Login.font

The font used to display the typed-in user name.

xlogin.Login.greeting
A string which identifies this window. The default is "Welcome to the X Window System".

xlogin.Login.unsecureGreeting
When X authorization is requested in the configuration file for this display and none is in use, this greeting replaces the standard greeting. It's default value is "This is an unsecure session"

xlogin.Login.greetFont
The font used to display the greeting.

xlogin.Login.greetColor
The color used to display the greeting.

xlogin.Login.namePrompt
The string displayed to prompt for a user name. *Xrdb* strips trailing white space from resource values, so to add spaces at the end of the prompt (usually a nice thing), add spaces escaped with backslashes. The default is "Login: "

xlogin.Login.passwdPrompt
The string displayed to prompt for a password. The default is "Password: ".

xlogin.Login.promptFont
The font used to display both prompts.

xlogin.Login.promptColor
The color used to display both prompts.

xlogin.Login.fail
A message which is displayed when the authentication fails. The default is "Login Failed, please try again".

xlogin.Login.failFont
The font used to display the failure message.

xlogin.Login.failColor
The color used to display the failure message.

xlogin.Login.failTimeout
The time (in seconds) that the fail message is displayed. The default is 30 seconds.

xlogin.Login.translations
This specifies the translations used for the login widget. Refer to the X Toolkit documentation for a complete discussion on translations. The default translation table is:

```
Ctrl<Key>H:          delete-previous-character() \n\
Ctrl<Key>D:          delete-character() \n\
Ctrl<Key>B:          move-backward-character() \n\
Ctrl<Key>F:          move-forward-character() \n\
Ctrl<Key>A:          move-to-begining() \n\
Ctrl<Key>E:          move-to-end() \n\
Ctrl<Key>K:          erase-to-end-of-line() \n\
Ctrl<Key>U:          erase-line() \n\
Ctrl<Key>X:          erase-line() \n\
Ctrl<Key>C:          restart-session() \n\
Ctrl<Key>\\:         abort-session() \n\
<Key>BackSpace:      delete-previous-character() \n\
<Key>Delete:         delete-previous-character() \n\
```

```
<Key>Return:        finish-field() \n\
<Key>:              insert-char() \
```

The actions which are supported by the widget are:

delete-previous-character
: Erases the character before the cursor.

delete-character
: Erases the character after the cursor.

move-backward-character
: Moves the cursor backward.

move-forward-character
: Moves the cursor forward.

move-to-begining
: (Apologies about the spelling error.) Moves the cursor to the beginning of the editable text.

move-to-end
: Moves the cursor to the end of the editable text.

erase-to-end-of-line
: Erases all text after the cursor.

erase-line
: Erases the entire text.

finish-field
: If the cursor is in the name field, proceeds to the password field; if the cursor is in the password field, check the current name/password pair. If the name/password pair are valid, *xdm* starts the session. Otherwise the failure message is displayed and the user is prompted to try again.

abort-session
: Terminates and restarts the server.

abort-display
: Terminates the server, disabling it. This is a rash action and is not accessible in the default configuration. It can be used to stop *xdm* when shutting the system down, or when using xdmshell.

restart-session
: Resets the X server and starts a new session. This can be used when the resources have been changed and you want to test them, or when the screen has been overwritten with system messages.

insert-char
: Inserts the character typed.

set-session-argument
: Specifies a single word argument which is passed to the session at startup. See the sections on **Xsession** and **Typical usage**.

allow-all-access
: Disables access control in the server, this can be used when the .Xauthority file cannot be created by xdm. Be very careful using this, it might be better to disconnect the machine from the network before doing this.

The Xstartup file

This file is typically a shell script. It is run as "root" and should be very careful about security. This is the place to put commands which make fake entries in /etc/utmp, mount users' home directories from file servers, display the message of the day, or abort the session if logins are not allowed. Various environment variables are set for the use of this script:

DISPLAY	is set to the associated display name
HOME	is set to the home directory of the user
USER	is set to the user name
PATH	is set to the value of **DisplayManager.DISPLAY.systemPath**
SHELL	is set to the value of **DisplayManager.DISPLAY.systemShell**
XAUTHORITY	may be set to an authority file

No arguments of any kind are passed to the script. *Xdm* waits until this script exits before starting the user session. If the exit value of this script is non-zero, *xdm* discontinues the session immediately and starts another authentication cycle.

The Xsession program

This is the command which is run as the user's session. It is run with the permissions of the authorized user, and has several environment variables specified:

DISPLAY	is set to the associated display name
HOME	is set to the home directory of the user
USER	is set to the user name
PATH	is set to the value of **DisplayManager.DISPLAY.userPath**
SHELL	is set to the user's default shell (from /etc/passwd)
XAUTHORITY	may be set to a non-standard authority file

At most installations, *Xsession* should look in $HOME for a file *.xsession* which would contain commands that each user would like to use as a session. This would replace the system default session. *Xsession* should also implement the system default session if no user-specified session exists. See the section **Typical Usage** below.

An argument may be passed to this program from the authentication widget using the 'set-session-argument' action. This can be used to select different styles of session. One very good use of this feature is to allow the user to escape from the ordinary session when it fails. This would allow users to repair their own *.xsession* if it fails, without requiring administrative intervention. The section on typical usage demonstrates this feature.

The Xreset file

Symmetrical with *Xstartup*, this script is run after the user session has terminated. Run as root, it should probably contain commands that undo the effects of commands in *Xstartup*, removing fake entries from */etc/utmp* or unmounting directories from file servers. The collection of environment variables that were passed to *Xstartup* are also given to *Xreset*.

Typical Usage

Actually, *xdm* is designed to operate in such a wide variety of environments that "typical" is probably a misnomer. However, this section will focus on making *xdm* a superior solution to traditional means of starting X from /etc/ttys or manually.

First off, the *xdm* configuration file should be set up. A good thing to do is to make a directory (*/usr/lib/X11/xdm* comes immediately to mind) which will contain all of the relevant files. Here is a reasonable configuration file, which could be named *xdm-config* :

```
DisplayManager.servers:                    /usr/lib/X11/xdm/Xservers
```

```
DisplayManager.errorLogFile:            /usr/lib/X11/xdm/xdm-errors
DisplayManager.pidFile:                 /usr/lib/X11/xdm/xdm-pid
DisplayManager*resources:               /usr/lib/X11/xdm/Xresources
DisplayManager*session:                 /usr/lib/X11/xdm/Xsession
DisplayManager._0.authorize:            true
DisplayManager*authorize:               false
```

As you can see, this file simply contains references to other files. Note that some of the resources are specified with "*" separating the components. These resources can be made unique for each different display, by replacing the "*" with the display-name, but normally this is not very useful. See the **Resources** section for a complete discussion.

The first file */usr/lib/X11/xdm/Xservers* contains the list of displays to manage. Most workstations have only one display, numbered 0, so the file will look like this:

```
:0 Local local /usr/bin/X11/X :0
```

This will keep */usr/bin/X11/X* running on this display and manage a continuous cycle of sessions.

The file */usr/lib/X11/xdm/xdm-errors* will contain error messages from *xdm* and anything output to stderr by *Xstartup, Xsession or Xreset*. When you have trouble getting *xdm* working, check this file to see if *xdm* has any clues to the trouble.

The next configuration entry, */usr/lib/X11/xdm/Xresources*, is loaded onto the display as a resource database using *xrdb (1)*. As the authentication widget reads this database before starting up, it usually contains parameters for that widget:

```
xlogin*login.translations: #override\
        <Key>F1: set-session-argument(failsafe) finish-field()\n\
        <Key>Return: set-session-argument() finish-field()
xlogin*borderWidth: 3
#ifdef COLOR
xlogin*greetColor: #f63
xlogin*failColor: red
xlogin*Foreground: black
xlogin*Background: #fdc
#else
xlogin*Foreground: black
xlogin*Background: white
#endif
```

The various colors specified here look reasonable on several of the displays we have, but may look awful on other monitors. As X does not currently have any standard color naming scheme, you might need to tune these entries to avoid disgusting results. Please note the translations entry; it specifies a few new translations for the widget which allow users to escape from the default session (and avoid troubles that may occur in it). Note that if #override is not specified, the default translations are removed and replaced by the new value, not a very useful result as some of the default translations are quite useful (like "<Key>: insert-char ()" which responds to normal typing).

The *Xstartup* file used here simply prevents login while the file */etc/nologin* exists. As there is no provision for displaying any messages here (there isn't any core X client which displays files), the user will probably be baffled by this behavior. I don't offer this as a complete example, but simply a demonstration of the available functionality.

Here is a sample *Xstartup* script:

```
#!/bin/sh
#
# Xstartup
#
# This program is run as root after the user is verified
#
if [ -f /etc/nologin ]; then
        exit 1
fi
exit 0
```

The most interesting script is *Xsession*. This version recognizes the special "failsafe" mode, specified in the translations in the *Xresources* file above, to provide an escape from the ordinary session:

```
#!/bin/sh
#
# Xsession
#
# This is the program that is run as the client
# for the display manager.  This example is
# quite friendly as it attempts to run a per-user
# .xsession file instead of forcing a particular
# session layout
#

case $# in
1)
        case $1 in
        failsafe)
                exec xterm -geometry 80x24-0-0 -ls
                ;;
        esac
esac

startup=$HOME/.xsession
resources=$HOME/.Xresources

if [ -f $startup ]; then
        exec $startup
        exec /bin/sh $startup
else
        if [ -f $resources ]; then
                xrdb -load $resources
        fi
        twm &
        exec xterm -geometry 80x24+10+10 -ls
fi
```

No *Xreset* script is necessary, so none is provided.

SOME OTHER POSSIBILITIES

You can also use *xdm* to run a single session at a time, using the 4.3 *init* options or other suitable daemon by specifying the server on the command line:

```
xdm -server ":0 SUN-3/60CG4 local /usr/bin/X :0"
```

Or, you might have a file server and a collection of X terminals. The configuration for this could look identical to the sample above, except the *Xservers* file might look like:

```
extol:0 VISUAL-19 foreign
exalt:0 NCD-19 foreign
explode:0 NCR-TOWERVIEW3000 foreign
```

This would direct *xdm* to manage sessions on all three of these terminals. See the section "Controlling Xdm" above for a description of using signals to enable and disable these terminals in a manner reminiscent of init(8).

One thing that *xdm* isn't very good at doing is coexisting with other window systems. To use multiple window systems on the same hardware, you'll probably be more interested in *xinit* .

SEE ALSO

X(1), xinit(1) and XDMCP

BUGS

COPYRIGHT

AUTHOR

Keith Packard, MIT X Consortium

NOTE

xdm is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xdpr - dump an X window directly to a printer

SYNOPSIS

xdpr [*filename*] [**-display** *host:display*] [**-P***printer*] [**-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.

-P*printer*

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

AUTHOR

Paul Boutin, MIT Project Athena
Michael R. Gretzinger, MIT Project Athena
Jim Gettys, MIT Project Athena

NOTE

xdpr is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xdpyinfo - display information utility for X

SYNOPSIS

xdpyinfo [-display *displayname*]

DESCRIPTION

Xdpyinfo 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 *xdpyinfo* when connected to display that supports an 8 plane screen and a 1 plane screen.

```
name of display:    :0.0
version number:    11.0
vendor string:    MIT X Consortium
vendor release number:    4
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
number of extensions:    4
    SHAPE
    MIT-SHM
    Multi-Buffering
    MIT-SUNDRY-NONSTANDARD
default screen number:    0
number of screens:    2

screen #0:
  dimensions:    1152x900 pixels (325x254 millimeters)
  resolution:    90x90 dots per inch
  depths (2):    1, 8
  root window id:    0x8006e
  depth of root window:    8 planes
  number of colormaps:    minimum 1, maximum 1
  default colormap:    0x8006b
  default number of colormap cells:    256
  preallocated pixels:    black 1, white 0
  options:    backing-store YES, save-unders YES
  current input event mask:    0xd0801d
    KeyPressMask            ButtonPressMask         ButtonReleaseMask
    EnterWindowMask         ExposureMask            SubstructureRedirectMask
    PropertyChangeMask      ColormapChangeMask
  number of visuals:    6
  default visual id:  0x80065
  visual:
    visual id:    0x80065
    class:    PseudoColor
```

```
    depth:    8 planes
    size of colormap:    256 entries
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x80066
    class:    DirectColor
    depth:    8 planes
    size of colormap:    8 entries
    red, green, blue masks:    0x7, 0x38, 0xc0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x80067
    class:    GrayScale
    depth:    8 planes
    size of colormap:    256 entries
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x80068
    class:    StaticGray
    depth:    8 planes
    size of colormap:    256 entries
    red, green, blue masks:    0x0, 0x0, 0x0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x80069
    class:    StaticColor
    depth:    8 planes
    size of colormap:    256 entries
    red, green, blue masks:    0x7, 0x38, 0xc0
    significant bits in color specification:    8 bits
  visual:
    visual id:    0x8006a
    class:    TrueColor
    depth:    8 planes
    size of colormap:    8 entries
    red, green, blue masks:    0x7, 0x38, 0xc0
    significant bits in color specification:    8 bits
  number of mono multibuffer types:    6
    visual id, max buffers, depth:    0x80065, 0, 8
    visual id, max buffers, depth:    0x80066, 0, 8
    visual id, max buffers, depth:    0x80067, 0, 8
    visual id, max buffers, depth:    0x80068, 0, 8
    visual id, max buffers, depth:    0x80069, 0, 8
    visual id, max buffers, depth:    0x8006a, 0, 8
  number of stereo multibuffer types:    0

screen #1:
  dimensions:    1152x900 pixels (325x254 millimeters)
  resolution:    90x90 dots per inch
  depths (1):    1
  root window id:    0x80070
```

```
depth of root window:    1 plane
number of colormaps:    minimum 1, maximum 1
default colormap:    0x8006c
default number of colormap cells:    2
preallocated pixels:    black 1, white 0
options:    backing-store YES, save-unders YES
current input event mask:    0xd0801d
  KeyPressMask            ButtonPressMask         ButtonReleaseMask
  EnterWindowMask         ExposureMask            SubstructureRedirectMask
  PropertyChangeMask      ColormapChangeMask
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
number of mono multibuffer types:    1
  visual id, max buffers, depth:    0x80064, 0, 1
number of stereo multibuffer types:    0
```

ENVIRONMENT

DISPLAY

To get the default host, display number, and screen.

SEE ALSO

X(1), xwininfo(1), xprop(1), xrdb(1)

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

xdpyinfo is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xedit - simple text editor for X

SYNTAX

xedit [*-toolkitoption* ...] [filename]

OPTIONS

Xedit accepts all of the standard X Toolkit command line options (see *X(1)*). The order of the command line options is not important.

filename Specifies the file that is to be loaded during start-up. This is the file which will be edited. If a file is not specified, *xedit* lets you load a file or create a new file after it has started up.

DESCRIPTION

Xedit provides a window consisting of the following four areas:

Commands Section	A set of commands that allow you to exit *xedit*, save the file, or load a new file into the edit window.
Message Window	Displays *xedit* messages. In addition, this window can be used as a scratch pad.
Filename Display	Displays the name of the file currently being edited, and whether this file is *Read - Write* or *Read Only*.
Edit Window	Displays the text of the file that you are editing or creating.

EDITING

The Athena Text widget is used for the three sections of this application that allow text input. The characters typed will go to the Text widget that the pointer cursor is currently over. If the pointer cursor is not over a text widget then the keypresses will have no effect on the application. This is also true for the special key sequences that popup dialog widgets, so typing Control-S in the filename widget will enable searching in that widget, not the edit widget.

Both the message window and the edit window will create a scrollbar if the text to display is too large to fit in that window. Horizontal scrolling is not allowed by default, but can be turned on through the Text widget's resources, see *Athena Widget Set* for the exact resource definition.

COMMANDS

Quit Quits the current editing session. If any changes have not been saved, *xedit* displays a warning message, allowing the user to save the file.

Save If file backups are enabled (see RESOURCES) xedit stores a copy of the original, unedited file in <prefix>*file*<suffix>, then overwrites the *file* with the contents of the edit window. The filename is retrieved from the Text widget directly to the right of the *Load* button.

Load Loads the file named in the text widget immediately to the right of the this button and displays it in the Edit window. If the currently displayed file has been modified a warning message will ask the user to save the changes, or press *load* again.

RESOURCES

For *xedit* the available resources are:

enableBackups (Class **EnableBackups**)

Specifies that, when edits made to an existing file are saved, *xedit* is to copy the original version of that file to <prefix>*file*<suffix> before it saves the changes. The default value for this resource is "off", stating that no backups should be created.

backupNamePrefix (Class **BackupNamePrefix**)

Specifies a string that is to be prepended to the backup filename. The default is that no string shall be prepended.

backupNameSuffix (Class **BackupNameSuffix**)
Specifies a string that is to be appended to the backup filename. The default is to append the string ".BAK".

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xedit*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Xedit xedit
        Paned paned
                Paned buttons
                        Command quit
                        Command save
                        Command load
                        Text filename
                Label bc_label
                Text messageWindow
                Label labelWindow
                Text editWindow
```

ENVIRONMENT

DISPLAY
to get the default host and display number.

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

FILES

/usr/lib/X11/app-defaults/Xedit - specifies required resources

SEE ALSO

X(1), xrdb(1), Athena Widget Set

RESTRICTIONS

There is no *undo* function.

COPYRIGHT

AUTHOR

Chris D. Peterson, MIT X Consortium

NOTE

xedit is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xev - print contents of X events

SYNOPSIS

xev [-display *displayname*] [-geometry *geom*]

DESCRIPTION

Xev creates a window and then asks the X server to send it notices called *events* whenever anything happens to the window (such as being moved, resized, typed in, clicked in, etc.). It is useful for seeing what causes events to occur and to display the information that they contain.

OPTIONS

-display *display*

This option specifies the X server to contact.

-geometry *geom*

This option specifies the size and/or location of the window.

SEE ALSO

X(1), xwininfo(1), xdpyinfo(1), Xlib Programmers Manual, X Protocol Specification

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

xev is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xeyes – watch over your shoulder

SYNOPSIS

xeyes [-option ...]

DESCRIPTION

Xeyes watches what you do and reports to the Boss.

OPTIONS

–fg *foreground color*
choose a different color for the pupil of the eyes.

–bg *background color*
choose a different color for the background.

–outline *outline color*
choose a different color for the outline of the eyes.

–center *center color*
choose a different color for the center of the eyes.

–backing { *WhenMapped Always NotUseful* }
selects an appropriate level of backing store.

–geometry *geometry*
define the initial window geometry; see *X(1)*.

–display *display*
specify the display to use; see *X(1)*.

–bd *border color*
choose a different color for the window border.

–bw *border width*
choose a different width for the window border.

–shape uses the SHAPE extension to shape the window.

SEE ALSO

X(1), X Toolkit documentation

COPYRIGHT

AUTHOR

Keith Packard, MIT X Consortium

NOTE

xeyes is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xfd - font displayer for X

SYNOPSIS

xfd [-options ...] -fn *fontname*

OPTIONS

Xfd accepts all of the standard toolkit command line options along with the additional options listed below:

–fn *font* This option specifies the font to be displayed.

–box This option indicates that a box outlining the area that would be filled with background color by an ImageText request.

–center This option indicates that each glyph should be centered in its grid.

–start *number*
This option specifies the glyph index of the upper left hand corner of the grid. This is used to view characters at arbitrary locations in the font. The default is 0.

–bc *color*
This option specifies the color to be used if ImageText boxes are drawn.

DESCRIPTION

The *xfd* utility creates a window containing the name of the font being displayed, a row of command buttons, several lines of text for displaying character metrics, and a grid containing one glyph per cell. 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. Each character glyph is drawn using the PolyText16 request (used by the *Xlib* routine **XDrawString16**). If the *-box* option is given, a rectangle will be drawn around each character, showing where an ImageText16 request (used by the *Xlib* routine **XDrawImageString16**) would cause background color to be displayed.

The origin of each glyph is normally set so that the character is drawn in the upper left hand corner of the grid cell. However, if a glyph has a negative left bearing or an unusually large ascent, descent, or right bearing (as is the case with *cursor* font), some character may not appear in their own grid cells. The *-center* option may be used to force all glyphs to be centered in their respective cells.

All the characters in the font may not fit in the window at once. To see the next page of glyphs, press the *Next* button at the top of the window. To see the previous page, press *Prev*. To exit *xfd*, press *Quit*.

Individual character metrics (index, width, bearings, ascent and descent) can be displayed at the top of the window by pressing on the desired character.

The font name displayed at the top of the window is the full name of the font, as determined by the server. See *xlsfonts* for ways to generate lists of fonts, as well as more detailed summaries of their metrics and properties.

X DEFAULTS

To be written.

SEE ALSO

X(1), xlsfonts(1), xrdb(1)

BUGS

The program should skip over pages full of non-existent characters.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium; previous program of the same name by Mark Lillibridge, MIT Project Athena.

NOTE

xfd is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xfontsel - point & click interface for selecting X11 font names

SYNTAX

xfontsel [*-toolkitoption ...*] [**-pattern** *fontname*] [**-print**] [**-sample** *text*]

DESCRIPTION

The *xfontsel* application provides a simple way to display the fonts known to your X server, examine samples of each, and retrieve the X Logical Font Description ("XLFD") full name for a font.

If **-pattern** is not specified, all fonts with XLFD 14-part names will be selectable. To work with only a subset of the fonts, specify **-pattern** followed by a partially or fully qualified font name; e.g., "-pattern *medium*" will select that subset of fonts which contain the string "medium" somewhere in their font name. Be careful about escaping wildcard characters in your shell.

If **-print** is specified on the command line the selected font specifier will be written to standard output when the *quit* button is activated. Regardless of whether or not **-print** was specified, the font specifier may be made the PRIMARY (text) selection by activating the *select* button.

The **-sample** option specifies the sample text to be used to display the selected font, overriding the default.

INTERACTIONS

Clicking any pointer button in one of the XLFD field names will pop up a menu of the currently-known possibilities for that field. If previous choices of other fields were made, only values for fonts which matched the previously selected fields will be selectable; to make other values selectable, you must deselect some other field(s) by choosing the "*" entry in that field. Unselectable values may be omitted from the menu entirely as a configuration option; see the **ShowUnselectable** resource, below. Whenever any change is made to a field value, *xfontsel* will assert ownership of the PRIMARY_FONT selection. Other applications (see, e.g., *xterm*) may then retrieve the selected font specification.

Clicking the left pointer button in the *select* widget will cause the currently selected font name to become the PRIMARY text selection as well as the PRIMARY_FONT selection. This then allows you to paste the string into other applications. The **select** button remains highlighted to remind you of this fact, and de-highlights when some other application takes the PRIMARY selection away. The *select* widget is a toggle; pressing it when it is highlighted will cause *xfontsel* to release the selection ownership and de-highlight the widget. Activating the *select* widget twice is the only way to cause *xfontsel* to release the PRIMARY_FONT selection.

RESOURCES

The application class is **XFontSel**. Most of the user-interface is configured in the app-defaults file; if this file is missing a warning message will be printed to standard output and the resulting window will be nearly incomprehensible.

Most of the significant parts of the widget hierarchy are documented in the app-defaults file (normally /usr/lib/X11/app-defaults/XFontSel).

Application specific resources:

cursor (class **Cursor**)
Specifies the cursor for the application window.

pattern (class **Pattern**)
Specifies the font name pattern for selecting a subset of available fonts. Equivalent to the **-pattern** option. Most useful patterns will contain at least one field delimiter; e.g. "*-m-*" for monospaced fonts.

printOnQuit (class **PrintOnQuit**)
If *True* the currently selected font name is printed to standard output when the quit

button is activated. Equivalent to the **-print** option.

Widget specific resources:

showUnselectable (class **ShowUnselectable**)
Specifies, for each field menu, whether or not to show values that are not currently selectable, based upon previous field selections. If shown, the unselectable values are clearly identified as such and do not highlight when the pointer is moved down the menu. The full name of this resource is **fieldN.menu.options.showUnselectable**, class **MenuButton.SimpleMenu.Options.ShowUnselectable**; where N is replaced with the field number (starting with the left-most field numbered 0). The default is True for all but field 11 (average width of characters in font) and False for field 11. If you never want to see unselectable entries, '*menu.options.showUnselectable:False' is a reasonable thing to specify in a resource file.

FILES

$XFILESEARCHPATH/XFontSel

SEE ALSO

xrdb(1)

BUGS

Sufficiently ambiguous patterns can be misinterpreted and lead to an initial selection string which may not correspond to what the user intended and which may cause the initial sample text output to fail to match the proffered string. Selecting any new field value will correct the sample output, though possibly resulting in no matching font.

Should be able to return a FONT for the PRIMARY selection, not just a STRING.

Any change in a field value will cause *xfontsel* to assert ownership of the PRIMARY_FONT selection. Perhaps this should be parameterized.

When running on a slow machine, it is possible for the user to request a field menu before the font names have been completely parsed. An error message indicating a missing menu is printed to stderr but otherwise nothing bad (or good) happens.

AUTHOR

Ralph R. Swick, Digital Equipment Corporation/MIT Project Athena

NOTE

xfontsel is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xgc - X graphics demo

SYNOPSIS

xgc [*-toolkitoption ...*]

DESCRIPTION

The *xgc* program demonstrates various features of the X graphics primitives. Try the buttons, see what they do; we haven't the time to document them, perhaps you do?

OPTIONS

Xgc accepts all of the standard X Toolkit command line options.

X DEFAULTS

This program accepts the usual defaults for toolkit applications.

ENVIRONMENT

DISPLAY

to get the default host and display number.

XENVIRONMENT

to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

SEE ALSO

X(1)

BUGS

This program isn't really finished yet.

COPYRIGHT

AUTHORS

Dan Schmidt, MIT

NOTE

xgc is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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).

AUTHORS

Bob Scheifler, MIT Laboratory for Computer Science,
Jim Gettys, MIT Project Athena (DEC).

NOTE

xhost is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xkill - kill a client by its X resource

SYNOPSIS

xkill [-display *displayname*] [-id *resource*] [-button number] [-frame] [-all]

DESCRIPTION

Xkill is a utility for forcing the X server to close connections to clients. This program is very dangerous, but is useful for aborting programs that have displayed undesired windows on a user's screen. If no resource identifier is given with *-id*, *xkill* will display a special cursor as a prompt for the user to select a window to be killed. If a pointer button is pressed over a non-root window, the server will close its connection to the client that created the window.

OPTIONS

-display *displayname*
This option specifies the name of the X server to contact.

-id *resource*
This option specifies the X identifier for the resource whose creator is to be aborted. If no resource is specified, *xkill* will display a special cursor with which you should select a window to be kill.

-button *number*
This option specifies the number of pointer button that should be used in selecting a window to kill. If the word "any" is specified, any button on the pointer may be used. By default, the first button in the pointer map (which is usually the leftmost button) is used.

-all This option indicates that all clients with top-level windows on the screen should be killed. *Xkill* will ask you to select the root window with each of the currently defined buttons to give you several chances to abort. Use of this option is highly discouraged.

-frame This option indicates that xkill should ignore the standard conventions for finding top-level client windows (which are typically nested inside a window manager window), and simply believe that you want to kill direct children of the root.

XDEFAULTS

Button Specifies a specific pointer button number or the word "any" to use when selecting windows.

SEE ALSO

X(1), xwininfo(1), XKillClient and XGetPointerMapping in the Xlib Programmers Manual, KillClient in the X Protocol Specification

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium
Dana Chee, Bellcore

NOTE

xkill is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xload - load average display for X

SYNOPSIS

xload [*-toolkitoption ...*] [-scale *integer*] [-update *seconds*] [-hl *color*] [-highlight *color*] [-jumpscroll *pixels*] [-label *string*] [-nolabel]

DESCRIPTION

The *xload* program displays a periodically updating histogram of the system load average.

OPTIONS

Xload accepts all of the standard X Toolkit command line options (see *X(1)*). The order of the options in unimportant. *xload also accepts the following additional options:*

-hl *color* **or -highlight** *color*

This option specifies the color of the scale lines.

-jumpscroll FPnumber of pixels

The number of pixels to shift the graph to the left when the graph reaches the right edge of the window. The default value is 1/2 the width of the current window. Smooth scrolling can be achieved by setting it to 1.

-label *string*

The string to put into the label above the load average.

-nolabel

If this command line option is specified then no label will be displayed above the load graph.

-scale *integer*

This option specifies the minimum number of tick marks in the histogram, where one division represents one load average point. If the load goes above this number, *xload* will create more divisions, but it will never use fewer than this number. The default is 1.

-update *seconds*

This option specifies the frequency in seconds at which *xload* updates its display. The minimum amount of time allowed between updates is 1 second (the default is 5 seconds).

RESOURCES

In addition to the resources available to each of the widgets used by *xload* there is one resource defined by the application itself.

showLabel (class **Boolean**)

If False then no label will be displayed.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xload*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XLoad xload
        Paned paned
                Label label
                StripChart load
```

ENVIRONMENT

DISPLAY

to get the default host and display number.

XENVIRONMENT

to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

FILES

/usr/lib/X11/app-defaults/XLoad - specifies required resources

SEE ALSO

X(1), xrdb(1), mem(4), Athena StripChart Widget.

BUGS

This program requires the ability to open and read the special system file */dev/kmem*. Sites that do not allow general access to this file should make *xload* belong to the same group as */dev/kmem* and turn on the *set group id* permission flag.

Reading /dev/kmem is inherently non-portable. Therefore, the routine used to read it (get_load.c) must be ported to each new operating system.

COPYRIGHT

AUTHORS

K. Shane Hartman (MIT-LCS) and Stuart A. Malone (MIT-LCS);
with features added by Jim Gettys (MIT-Athena), Bob Scheifler (MIT-LCS), Tony Della Fera (MIT-Athena), and Chris Peterson (MIT-LCS).

NOTE

xload is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xlogo - X Window System logo

SYNOPSIS

xlogo [*-toolkitoption ...*]

DESCRIPTION

The *xlogo* program displays the X Window System logo. This program is nothing more than a wrapper around the *undocumented* Athena Logo widget.

OPTIONS

Xlogo accepts all of the standard X Toolkit command line options, see *X(1)* for details.

RESOURCES

This program uses the *Logo* widget in the Athena widget set. It understands all of the core resource names and classes as well as:

width (class **Width**)
Specifies the width of the logo. The default width is 100 pixels.

height (class **Height**)
Specifies the height of the logo. The default height is 100 pixels.

foreground (class **Foreground**)
Specifies the color for the logo. The default is depends on whether *reverseVideo* is specified. If *reverseVideo* is specified the default is *lwhite*, otherwise the default is *black*.

reverseVideo (class **ReverseVideo**)
This option indicates that reverse video should be simulated, changing the default value for the foreground, background and border colors.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xlogo*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
XLogo xlogo
    Logo xlogo
```

ENVIRONMENT

DISPLAY
to get the default host and display number.

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

FILES

/usr/lib/X11/app-defaults/XLogo - specifies required resources

SEE ALSO

X(1), xrdb(1)

COPYRIGHT

AUTHORS

Ollie Jones of Apollo Computer and Jim Fulton of the MIT X Consortium wrote the logo graphics routine, based on a graphic design by Danny Chong and Ross Chapman of Apollo Computer.

NOTE

xlogo is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xlsatoms - list interned atoms defined on server

SYNOPSIS

xlsatoms [-options ...]

DESCRIPTION

Xlsatoms lists the interned atoms. By default, all atoms starting from 1 (the lowest atom value defined by the protocol) are listed until unknown atom is found. If an explicit range is given, *xlsatoms* will try all atoms in the range, regardless of whether or not any are undefined.

OPTIONS

–display *dpy*

This option specifies the X server to which to connect.

–format *string*

This option specifies a *printf*-style string used to list each atom *<value,name>* pair, printed in that order (*value* is an *unsigned long* and *name* is a *char **). *Xlsatoms* will supply a newline at the end of each line. The default is *%ld\t%s*.

–range *[low]-[high]*

This option specifies the range of atom values to check. If *low* is not given, a value of 1 assumed. If *high* is not given, *xlsatoms* will stop at the first undefined atom at or above *low*.

–name *string*

This option specifies the name of an atom to list. If the atom does not exist, a message will be printed on the standard error.

SEE ALSO

X(1), Xserver(1), xprop(1)

ENVIRONMENT

DISPLAY

to get the default host and display to use.

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

xlsatoms is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xlsclients - list client applications running on a display

SYNOPSIS

xlsclients [-display *displayname*] [-a] [-l] [-m maxcmdlen]

DESCRIPTION

Xlsclients is a utility for listing information about the client applications running on a display. It may be used to generate scripts representing a snapshot of the the user's current session.

OPTIONS

–display *displayname*

This option specifies the X server to contact.

–a This option indicates that clients on all screens should be listed. By default, only those clients on the default screen are listed.

–l This option indicates that a long listing showing the window name, icon name, and class hints in addition to the machine name and command string shown in the default listing.

–m *maxcmdlen*

This option specifies the maximum number of characters in a command to print out. The default is 1000.

ENVIRONMENT

DISPLAY

To get the default host, display number, and screen.

SEE ALSO

X(1), xwininfo(1), xprop(1)

COPYRIGHT

AUTHOR

Jim Fulton, MIT X Consortium

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[**l**[**l**]] This option indicates that medium, long, and very long listings, respectively, should be generated for each font.

-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-preemptable servers, not with this program.

AUTHOR

Mark Lillibridge, MIT Project Athena; Jim Fulton, MIT X Consortium; Phil Karlton, SGI

NOTE

xlsfonts is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

xlswins is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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 *colororpixelvalue*

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

AUTHOR

Jim Fulton, MIT X Consortium

NOTE

xmag is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xman - Manual page display program for the X Window System.

SYNOPSIS

xman [-options ...]

DESCRIPTION

Xman is a manual page browser. The default size of the initial *xman* window is small so that you can leave it running throughout your entire login session. In the initial window there are three options: *Help* will pop up a window with on-line help, *Quit* will exit, and *Manual Page* will pop up a window with a manual page browser in it. You may pop up more than one manual page browser window from a single execution of *xman*.

For further information on using *xman* please read the on-line help information. The rest of this manual page will discuss customization of *xman*.

CUSTOMIZING XMAN

Xman allows customization of both the directories to be searched for manual pages, and the name that each directory will map to in the *Sections* menu. Xman determines which directories it will search by reading the *MANPATH* environment variable. If no *MANPATH* is found then the directory is /usr/man is searched on POSIX systems. This environment is expected to be a colon-separated list of directories for xman to search.

setenv MANPATH /mit/kit/man:/usr/man

By default, *xman* will search each of the following directories (in each of the directories specified in the users MANPATH) for manual pages. If manual pages exist in that directory then they are added to list of manual pages for the corresponding menu item. A menu item is only displayed for those sections that actually contain manual pages.

Directory	Section Name
man1	(1) User Commands
man2	(2) System Calls
man3	(3) Subroutines
man4	(4) Devices
man5	(5) File Formats
man6	(6) Games
man7	(7) Miscellaneous
man8	(8) Sys. Administration
manl	(l) Local
mann	(n) New
mano	(o) Old

For instance, a user has three directories in her manual path and each contain a directory called *man8*. All these manual pages will appear alphabetically sorted when the user selects the menu item called *(8) Subroutines*. If there is no directory called *mano* in any of the directories in her MANPATH, or there are no manual pages in any of the directories called *mano* then no menu item will be displayed for the section called *(o) Old*.

By using the *mandesc* file a user or system manager is able to more closely control which manual pages will appear in each of the sections represented by menu items in the *Sections* menu. This functionality is only available on a section by section basis, and individual manual pages may not be handled in this manner (Although generous use of symbolic links - ln(1) - will allow almost any configuration you can imagine).

The format of the mandesc file is a character followed by a label. The character determines which of the sections will be added under this label. For instance suppose that you would like to create an extra menu item that contains all programmer subroutines. This label should contain all manual pages in both sections two and three. The *mandesc* file would look like this:

2Programmer Subroutines
3Programmer Subroutines

This will add a menu item to the *Sections* menu that would bring up a listing of all manual pages in sections two and three of the Programmers Manual. Since the label names are *exactly* the same they will be added to the same section. Note, however, that the original sections still exist.

If you want to completely ignore the default sections in a manual directory then add the line:

no default sections

anywhere in your mandesc file. This keeps xman from searching the default manual sections *In that directory only.* As an example, suppose you want to do the same thing as above, but you don't think that it is useful to have the *System Calls* or *Subroutines* sections any longer. You would need to duplicate the default entries, as well as adding your new one.

no default sections
1(1) User Commands
2Programmer Subroutines
3Programmer Subroutines
4(4) Devices
5(5) File Formats
6(6) Games
7(7) Miscellaneous
8(8) Sys. Administration
l(l) Local
n(n) New
o(o) Old

Xman will read any section that is of the from *man<character>*, where <character> is an upper or lower case letter (they are treated distinctly) or a numeral (0-9). Be warned, however, that man(1) and catman(8) will not search directories that are non-standard.

COMMAND LINE OPTIONS

Xman supports all standard Toolkit command line arguments (see *X(1)*). The following additional arguments are supported.

-helpfile *filename*
Specifies a helpfile to use other than the default.

-bothshown
Allows both the manual page and manual directory to be on the screen at the same time.

-notopbox
Starts without the Top Menu with the three buttons in it.

-geometry *WxH+X+Y*
Sets the size and location of the Top Menu with the three buttons in it.

-pagesize *WxH+X+Y*
Sets the size and location of all the Manual Pages.

WIDGETS

In order to specify resources, it is useful to know the hierarchy of the widgets which compose *xman*. In the notation below, indentation indicates hierarchical structure. The widget class name is given first, followed by the widget instance name.

```
Xman xman      (This widget is never used)
        TopLevelShell  topbox
                Form  form
                        Label  topLabel
                        Command  helpButton
                        Command  quitButton
                        Command  manpageButton
                TransientShell  search
                        DialogWidgetClass  dialog
                                Label  label
                                Text  value
                                Command  manualPage
                                Command  apropos
                                Command  cancel
                TransientShell  pleaseStandBy
                        Label  label
        TopLevelShell  manualBrowser
                Paned  Manpage_Vpane
                        Paned  horizPane
                                MenuButton  options
                                MenuButton  sections
                                Label  manualBrowser
                        Viewport  directory
                                List  directory
                                List  directory
                                .
                                . (one for each section,
                                .  created "on the fly")
                                .
                        ScrollByLine  manualPage
                SimpleMenu  optionMenu
                        SmeBSB  displayDirectory
                        SmeBSB  displayManualPage
                        SmeBSB  help
                        SmeBSB  search
                        SmeBSB  showBothScreens
                        SmeBSB  removeThisManpage
                        SmeBSB  openNewManpage
                        SmeBSB  showVersion
                        SmeBSB  quit
                SimpleMenu  sectionMenu
                        SmeBSB  <name of section>
                                .
                                . (one for each section)
                                .
                TransientShell  search
                        DialogWidgetClass  dialog
                                Label  label
```

```
                        Text value
                        Command manualPage
                        Command apropos
                        Command cancel
            TransientShell pleaseStandBy
                    Label label
            TransientShell likeToSave
                    Dialog dialog
                        Label label
                        Text value
                        Command yes
                        Command no
    TopLevelShell help
            Paned Manpage_Vpane
                    Paned horizPane
                        MenuButton options
                        MenuButton sections
                        Label manualBrowser
                    ScrollByLine manualPage
            SimpleMenu optionMenu
                    SmeBSB displayDirectory
                    SmeBSB displayManualPage
                    SmeBSB help
                    SmeBSB search
                    SmeBSB showBothScreens
                    SmeBSB removeThisManpage
                    SmeBSB openNewManpage
                    SmeBSB showVersion
                    SmeBSB quit
```

APPLICATION RESOURCES

xman has the following application-specific resources which allow customizations unique to *xman*.

manualFontNormal (Class **Font**)
The font to use for normal text in the manual pages.

manualFontBold (Class **Font**)
The font to use for bold text in the manual pages.

manualFontItalic (Class **Font**)
The font to use for italic text in the manual pages.

directoryFontNormal (Class **Font**)
The font to use for the directory text.

bothShown (Class **Boolean**)
Either 'true' or 'false', specifies whether or not you want both the directory and the manual page shown at start up.

directoryHeight (Class **DirectoryHeight**)
The height in pixels of the directory, when the directory and the manual page are shown simultaneously.

topCursor (Class **Cursor**)
The cursor to use in the top box.

helpCursor (Class **Cursor**)
The cursor to use in the help window.

manpageCursor (Class **Cursor**)
The cursor to use in the manual page window.

searchEntryCursor (Class **Cursor**)
The cursor to use in the search entry text widget.

pointerColor (Class **Foreground**)
This is the color of all the cursors (pointers) specified above. The name was chosen to be compatible with xterm.

helpFile (Class **File**)
Use this rather than the system default helpfile.

topBox (Class **Boolean**)
Either 'true' or 'false', determines whether the top box (containing the help, quit and manual page buttons) or a manual page is put on the screen at start-up. The default is true.

verticalList (Class **Boolean**)
Either 'true' or 'false', determines whether the directory listing is vertically or horizontally organized. The default is horizontal (false).

GLOBAL ACTIONS

Xman defines all user interaction through global actions. This allows the user to modify the translation table of any widget, and bind any event to the new user action. The list of actions supported by *xman* are:

GotoPage(*page***)** When used in a manual page display window this will allow the user to move between a directory and manual page display. The *page* argument can be either **Directory** or **ManualPage**.

Quit() This action may be used anywhere, and will exit xman.

Search(*type***,** *action***)** Only useful when used in a search popup, this action will cause the search widget to perform the named search type on the string in the search popup's value widget. This action will also pop down the search widget. The *type* argument can be either **Apropos**, **Manpage** or **Cancel**. If an *action* of **Open** is specified then xman will open a new manual page to display the results of the search, otherwise xman will attempt to display the results in the parent of the search popup.

PopupHelp() This action may be used anywhere, and will popup the help widget.

PopupSearch() This action may be used anywhere except in a help window. It will cause the search popup to become active and visible on the screen, allowing the user search for a manual page.

CreateNewManpage()
This action may be used anywhere, and will create a new manual page display window.

RemoveThisManpage()
This action may be used in any manual page or help display window. When called it will remove the window, and clean up all resources associated with it.

SaveFormattedPage(*action***)**
This action can only be used in the "likeToSave" popup widget, and tells xman whether to **Save** or **Cancel** a save of the manual page that has just been formatted.

ShowVersion() This action may be called from any manual page or help display window,

and will cause the informational display line to show the current version of xman.

FILES

<manpath directory>/man*<character>*	
<manpath directory>/cat*<character>*	
<manpath directory>/mandesc	
/usr/lib/X11/app-defaults/Xman	specifies required resources
/tmp	*Xman* creates temporary files in /tmp for all unformatted man pages and all apropos searches.

SEE ALSO

X(1), X(8C), man(1), apropos(1), catman(8), Athena Widget Set

ENVIRONMENT

DISPLAY	the default host and display to use.
MANPATH	the search path for manual pages. Directories are separated by colons (e.g. /usr/man:/mit/kit/man:/foo/bar/man).
XENVIRONMENT	to get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.
XAPPLRESDIR	A string that will have "Xman" appended to it. This string will be the full path name of a user app-defaults file to be merged into the resource database after the system app-defaults file, and before the resources that are attached to the display.

BUGS

There probably are some.

COPYRIGHT

AUTHORS

Chris Peterson, MIT X Consortium from the V10 version written by Barry Shein formerly of Boston University.

NOTE

xman is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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 *ttyModes* 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*ttyModes: erase ^?" | xrdb -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

AUTHOR

Jim Fulton, MIT X Consortium, rewritten from an earlier version by David Rosenthal of Sun Microsystems.

NOTE

xmodmap is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xpostage - postage meter for X11

SYNOPSIS

xpostage [-*toolkitoption* ...] [-option ...]

DESCRIPTION

Xpostage is an X11 application that monitors your mailbox and displays the number of mail messages it contains. A mouse button 1 or 2 click on the *xpostage* meter allows you to review the 'envelopes' of each piece of mail that you have received, from the most recent message to the oldest (64 maximum). The 'envelopes' show the 'To' line, the 'From' line, the 'Date' line, the 'Subject' line and as much of the mail text that will fit. The 'envelope' is implemented as a readonly textwidget. This allows text to be selected (cut) using the mouse for use in another X client. The 'next' and 'previous' buttons let you page thru your mail (only the last 64 messages are available). The first message shown is the latest mail to arrive, so the 'previous' button is used to go backward chronologically. The 'next' button goes forward chronologically after you have paged backwards. These buttons will beep when there is no more mail to read in the selected direction. The 'quit' button will return *xpostage* to its original state.

Xpostage announces the arrival of new mail by changing the counter and sounding the bell. The bell can optionally be disabled. There is also a 'visual bell' which can be enabled instead of or in addition to the audible bell. This causes the postage meter to reverse colors when new mail arrives. The colors return to normal when mail is scanned with *xpostage*, when the counter decreases (mail is read and delete) or by 'clicking' mouse button 3 on the postage meter.

OPTIONS

Xpostage accepts all of the standard X Toolkit command line options along with the additional options listed below:

-checktime *seconds*
-ct *seconds*
Sets the number of seconds between mail checks. (Defaults to 30 seconds.)

-lines *lines*
-ln *lines*
Sets the number of lines displayed in the text widget. The default (and minimum) is 4 lines to hold the standard header lines.

-mailbox *mailbox*
-mb *mailbox*
Sets the mailbox to be monitored. (Defaults to /usr/spool/mail/$USER.)

-nobell
-nb Disables the audible bell.

-visualbell
-vb Enables the visual bell.

WIDGET USAGE

Xpostage uses several widget types to construct the objects it uses.

The 'postage meter' is made up of a Postage widget called "postage".

The 'envelope' is a pop-up shell called "Envelope", containing a Form widget called "form" which contains a Text widget called "text", and three Command widgets called "next", "previous", and "quit".

X DEFAULTS

Xpostage understands all of the core X Toolkit resource names and classes as well as those listed below, which are all of class **XPostage.**

.checkTime

Specifies the time in seconds to wait between mailbox checks.

.lines Specifies the number of lines to display in the text widget. The default (and minimum) is 4 lines to hold the standard header lines.

.mailbox
Specifies the mailbox to monitor. The default is /usr/spool/mail/$USER.

.noBell Disables the audible bell sounded upon receipt of new mail. The default is **false**.

.visualBell
Enables the visual bell displayed upon receipt of new mail. The default is **false**.

The following are the recommend constructions for setting resources:

*XPostage.*resource*

*XPostage.postage**resource*

*XPostage.Envelope**resource*

SEE ALSO

X(1), xrdb(1)

BUGS

A bug in the Athena textwidget causes text to be invisible when the toolkit reverse video option is used. Everything seems acceptable if the background is set to 'black' and the foreground is set to 'white' using toolkit options rather than using the reverse video option.

AUTHOR

Cliff Herod, Convex Technical Assistance Center

NOTES

xpostage is an optional product; for more information contact your CONVEX sales representative.

NAME

xpr – print an X window dump

SYNOPSIS

xpr [**–device** *dev*] [**–scale** *scale*] [**–height** *inches*] [**–width** *inches*] [**–left** *inches*] [**–top** *inches*] [**–header** *string*] [**–trailer** *string*] [**–landscape**] [**–portrait**] [**–plane** *number*] [**–gray**] [**–rv**] [**–compact**] [**–output** *filename*] [**–append** *filename*] [**–noff**] [**–split** *n*] [**–psfig**] [**–density** *dpi*] [**–cutoff** *level*] [**–noposition**] [**–gamma** *correction*] [**–render** *algorithm*] [**–slide**] [*filename*]

DESCRIPTION

xpr takes as input a window dump file produced by *xwd(1)* and formats it for output on PostScript printers, the Digital LN03 or LA100, the IBM PP3812 page printer, the HP LaserJet (or other PCL printers), or the HP PaintJet. 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

–device *dev*

Specifies the device on which the file will be printed. Currently supported:

la100	Digital LA100
ljet	HP LaserJet series and other monochrome PCL devices such as ThinkJet, QuietJet, RuggedWriter, HP2560 series, and HP2930 series printers
ln03	Digital LN03
pjet	HP PaintJet (color mode)
pjetxl	HP HP PaintJet XL Color Graphics Printer (color mode)
pp	IBM PP3812
ps	PostScript printer

The default device is the *LN03*, for historical reasons. **-device lw** (LaserWriter) is equivalent to -device ps and is provided only for backwards compatibility.

–scale *scale*

Affects the size of the window on the page. The PostScript, LN03, and HP 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 page.

–width *inches*

Specifies the maximum width of the page.

–left *inches*

Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

–top *inches*

Specifies the top margin for the picture in inches. Fractions are allowed.

–header *string*

Specifies a header string to be printed above the window.

–trailer *string*

Specifies a trailer string to be printed below the window.

-landscape
Forces the window to printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

-plane *number*
Specifies which bit plane to use in an image. The default is to use the entire image and map values into black and white based on color intensities.

-gray *2 | 3 | 4*
Uses a simple 2x2, 3x3, or 4x4 gray scale conversion on a color image, rather than mapping to strictly black and white. This doubles, triples, or quadruples the effective width and height of the image.

-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.

-psfig Suppress translation of the PostScript picture to the center of the page.

-density *dpi*
Indicates what dot-per-inch density should be used by the HP printer.

-cutoff *level*
Changes the intensity level where colors are mapped to either black or white for monochrome output on a LaserJet printer. The *level* is expressed as percentage of full brightness. Fractions are allowed.

-noposition
This option causes header, trailer, and image positioning command generation to be bypassed for LaserJet, PaintJet and PaintJet XL printers.

-gamma *correction*
This changes the intensity of the colors printed by PaintJet XL printer. The *correction* is a floating point value in the range 0.00 to 3.00. Consult the operator's manual to determine the correct value for the specific printer.

-render *algorithm*
This allows PaintJet XL printer to render the image with the best quality versus performance tradeoff. Consult the operator's manual to determine which *algorithms* are available.

-slide This option allows overhead transparencies to be printed using the PaintJet and PaintJet

XL printers.

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 from panel. There is no easy solution to this problem. It will probably be necessary to either cut and paste, or to rework the application to produce a less complex picture.

There are several limitations on the LA100 support: the picture 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.

For color images, should map directly to PostScript image support.

HP PRINTERS

If no **–density** is specified on the command line 300 dots per inch will be assumed for *ljet* and 90 dots per inch for *pjet*. Allowable *density* values for a LaserJet printer are 300, 150, 100, and 75 dots per inch. Consult the operator's manual to determine densities supported by other printers.

If no **–scale** is specified the image will be expanded to fit the printable page area.

The default printable page area is 8x10.5 inches. Other paper sizes can be accomadated using the **–height** and **–width** options.

Note that a 1024x768 image fits the default printable area when processed at 100 dpi with scale=1, the same image can also be printed using 300 dpi with scale=3 but will require considerably more data be transferred to the printer.

xpr may be tailored for use with monochrome PCL printers other than the LaserJet. To print on a ThinkJet (HP2225A) *xpr* could be invoked as:

xpr -density 96 -width 6.667 *filename*

or for black-and-white output to a PaintJet:

xpr -density 180 *filename*

The monochrome intensity of a pixel is computed as 0.30*R + 0.59*G + 0.11*B. If a pixel's computed intensity is less than the **-cutoff** level it will print as white. This maps light-on-dark display images to black-on-white hardcopy. The default cutoff intensity is 50% of full brightness. Example: specifying **-cutoff 87.5** moves the white/black intensity point to 87.5% of full brightness.

A LaserJet printer must be configured with sufficient memory to handle the image. For a full page at 300 dots per inch approximately 2MB of printer memory is required.

Color images are produced on the PaintJet at 90 dots per inch. The PaintJet is limited to sixteen colors from its 330 color palette on each horizontal print line. *xpr* will issue a warning message if more than sixteen colors are encountered on a line. *xpr* will program the PaintJet for the first sixteen colors encountered on each line and use the nearest matching programmed value for other colors present on the line.

Specifying the **-rv**, reverse video, option for the PaintJet will cause black and white to be interchanged on the output image. No other colors are changed.

Multiplane images must be recorded by *xwd* in *ZPixmap* format. Single plane (monochrome) images may be in either *XYPixmap* or *ZPixmap* format.

Some PCL printers do not recognize image positioning commands. Output for these printers will not be centered on the page and header and trailer strings may not appear where expected.

The **-gamma** and **-render** options are supported only on the PaintJet XL printers.

The **-slide** option is not supported for LaserJet printers.

The **-split** option is not supported for HP printers.

COPYRIGHT

AUTHORS

Michael R. Gretzinger, MIT Project Athena, Jose Capo, MIT Project Athena (PP3812 support), Marvin Solomon, University of Wisconsin, Bob Scheifler, MIT, Angela Bock and E. Mike Durbin, Rich Inc. (grayscale), Larry Rupp, HP (HP printer support).

NOTE

xpr is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xprop - property displayer for X

SYNOPSIS

xprop [-help] [-grammar] [-id *id*] [-root] [-name *name*] [-frame] [-font *font*] [-display *display*] [-len *n*] [-notype] [-fs *file*] [-remove *property-name*] [-spy] [-f *atom format* [*dformat*]]* [*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.

-frame Specifies that when selecting a window by hand (i.e. if none of **-name**, **-root**, or **-id** are given), look at the window manager frame (if any) instead of looking for the client window.

-remove *property-name*
Specifies the name of a property to be removed from the indicated window.

-spy Examine window properties forever, looking for property change events.

-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 *format*s and *dformat*s. However, for the vast majority of users and uses, this should not be necessary as the built in defaults contain the *format*s and *dformat*s necessary to display all the standard properties. It should only be necessary to specify *format*s and *dformat*s 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* | m*n*

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*. m*n* 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

AUTHOR

Mark Lillibridge, MIT Project Athena

NOTE

xprop is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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:

BITS_PER_RGB=num
: the number of significant bits in an RGB color specification. This is the log base 2 of the number of distinct shades of each primary that the hardware can generate. Note that it usually is not related to PLANES.

CLASS=visualclass
: one of StaticGray, GrayScale, StaticColor, PseudoColor, TrueColor, DirectColor. This is the visual class of the root window of the default screen.

COLOR
: defined only if CLASS is one of StaticColor, PseudoColor, TrueColor, or DirectColor.

HEIGHT=num
: the height of the default screen in pixels.

SERVERHOST=hostname
: the hostname portion of the display to which you are connected.

HOST=hostname
: the same as **SERVERHOST**.

CLIENTHOST=hostname
: the name of the host on which *xrdb* is running.

PLANES=num
: the number of bit planes (the depth) of the root window of the default screen.

RELEASE=num
: the vendor release number for the server. The interpretation of this number will vary depending on VENDOR.

REVISION=num
: the X protocol minor version supported by this server (currently 0).

VERSION=num
: the X protocol major version supported by this server (should always be 11).

VENDOR=vendor
: a string specifying the vendor of the server.

WIDTH=num
the width of the default screen in pixels.

X_RESOLUTION=num
the x resolution of the default screen in pixels per meter.

Y_RESOLUTION=num
the y resolution of the default screen in pixels per meter.

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 was 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. Note that this option does a lexicographic sorted merge of the two inputs, which is almost certainly not what you want, but remains for backward compatibility.

–remove
This option indicates that the RESOURCE_MANAGER property should be removed from its window.

–retain This option indicates that the server should be instructed not to reset if *xrdb* is the first client.

–edit *filename*
This option indicates that the contents of the RESOURCE_MANAGER property should be edited into the given file, replacing any values already listed there. This allows you to put changes that you have made to your defaults back into your resource file, preserving any comments or preprocessor lines.

–backup *string*
This option specifies a suffix to be appended to the filename used with **–edit** to generate a backup file.

–D*name*[=*value*]
This option is passed through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef*.

–U*name* This option is passed through to the preprocessor and is used to remove any definitions of this symbol.

–I*directory*
This option is passed through to the preprocessor and is used to specify a directory to search for files that are referenced with *#include*.

FILES
Generalizes ~/*.Xdefaults* files.

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

AUTHORS
Phil Karlton, rewritten from the original by Jim Gettys

NOTE
xrdb is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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

AUTHORS

Jim Gettys, Digital Equipment Corp., MIT Project Athena

NOTE

xrefresh is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xset - user preference utility for X

SYNOPSIS

xset [-display *display*] [-b] [b on/off] [b [*volume* [*pitch* [*duration*]]]] [[-]bc] [-c] [c on/off] [c [*volume*]] [[-+]fp[-+=] *path*[,*path*[,...]]] [fp default] [fp rehash] [[-]led [*integer*]] [led on/off] [m[ouse] [*acceleration* [*threshold*]]] [m[ouse] 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.

bc The **bc** option controls *bug compatibility* mode in the server, if possible; a preceding dash(-) disables the mode, otherwise the mode is enabled. Various pre-R4 clients pass illegal values in some protocol requests, and pre-R4 servers did not correctly generate errors in these cases. Such clients, when run against an R4 server, will terminate abnormally or otherwise fail operate correctly. Bug compatibility mode explicitly reintroduces certain bugs into the X server, so that many such clients can still be run. This mode should be used with care; new application development should be done with this mode disabled. The server must support the MIT-SUNDRY-NONSTANDARD protocol extension in order for this option to work.

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), xrdb(1), xsetroot(1)

COPYRIGHT

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science
David Krikorian, MIT Project Athena (X11 version)

NOTE

xset is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xsetroot – root window parameter setting utility for X

SYNOPSIS

xsetroot [-help] [-def] [-display *display*] [-cursor *cursorfile maskfile*] [-cursor_name *cursorname*] [-bitmap *filename*] [-mod *x y*] [-gray] [-grey] [-fg *color*] [-bg *color*] [-rv] [-solid *color*] [-name *string*]

DESCRIPTION

The *setroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a personalized look that you like, then put the *xsetroot* command that produces it into your X startup file. If no options are specified, or if *-def* is specified, the window is reset to its default state. The *-def* option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (-solid, -gray, -grey, -bitmap, and -mod) may be specified at a time.

OPTIONS

The various options are as follows:

-help Print a usage message and exit.

-def Reset unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*
This lets you change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be 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.

-cursor_name *cursorname*
This lets you change the pointer cursor to one of the standard cursors from the cursor font. Refer to appendix B of the X protocol for the names (except that the XC_ prefix is elided for this option).

-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*
This sets the background of the root window to the specified color. This option is only

useful on color servers.

-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)

AUTHOR
Mark Lillibridge, MIT Project Athena

NOTE
xsetroot is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xstdcmap - X standard colormap utility

SYNOPSIS

xstdcmap [-all] [-best] [-blue] [-default] [-delete *map*] [-display *display*] [-gray] [-green] [-help] [-red] [-verbose]

DESCRIPTION

The *xstdcmap* utility can be used to selectively define standard colormap properties. It is intended to be run from a user's X startup script to create standard colormap definitions in order to facilitate sharing of scarce colormap resources among clients. Where at all possible, colormaps are created with read-only allocations.

OPTIONS

The following options may be used with *xstdcmap*:

–all This option indicates that all six standard colormap properties should be defined on each screen of the display. Not all screens will support visuals under which all six standard colormap properties are meaningful. *xstdcmap* will determine the best allocations and visuals for the colormap properties of a screen. Any previously existing standard colormap properties will be replaced.

–best This option indicates that the RGB_BEST_MAP should be defined.

–blue This option indicates that the RGB_BLUE_MAP should be defined.

–default
This option indicates that the RGB_DEFAULT_MAP should be defined.

–delete *map*
This option specifies that a standard colormap property should be removed. *mapP may be one of: default, best, red, green, blue, or gray.*

–display *display*
This option specifies the host and display to use; see *X(1)*.

–gray This option indicates that the RGB_GRAY_MAP should be defined.

–green This option indicates that the RGB_GREEN_MAP should be defined.

–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 *xstdcmap*.

–red This option indicates that the RGB_RED_MAP should be defined.

–verbose
This option indicates that *xstdcmap* should print logging information as it parses its input and defines the standard colormap properties.

ENVIRONMENT

DISPLAY
to get default host and display number.

SEE ALSO

X(1)

COPYRIGHT

AUTHOR

Donna Converse, MIT X Consortium

NOTE

xstdcmap is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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 as well as the following (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.

–cn This option indicates that newlines should not be cut in line-mode selections.

+cn This option indicates that newlines should be cut in line-mode selections.

–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 a 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 is to do overstriking of the normal font.

–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 the 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 a 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.

–mc milliseconds
This option specifies the maximum time between multi-click selections.

–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.

-tm *string*
This option specifies a series of terminal setting keywords followed by the characters that should be bound to those functions, similar to the *stty* program. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete.

-tn *name*
This option specifies the name of the terminal type to be set in the TERM environment variable. This terminal type must exist in the *termcap(5)* database and should have *li#* and *co#* entries.

-ut This option indicates that *xterm* shouldn't write a record into the the system log file */etc/utmp*.

+ut This option indicates that *xterm* should write a record into the system log file */etc/utmp*.

-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.

–wf This option indicates that *xterm* should wait for the window to be mapped the first time before starting the subprocess so that the initial terminal size settings and environment variables are correct. It the application's responsibility to catch subsequent terminal size changes.

+wf This option indicates that *xterm* show not wait before starting the subprocess.

–C This option indicates that this window should receive console output. This is not supported on all systems.

–S*ccn* This option specifies the last two letters of the name of a pseudoterminal to use in slave mode, plus the number of the inherited file descriptor. The option is parsed "%c%c%d". This 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**.

–n *string*
This option specifies the icon name for *xterm*'s windows. It is shorthand for specifying the "**iconName*" resource. Note that this is not the same as the toolkit option **-name** (see below). The default icon name is the application name.

–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 *fixed.*

–name *name*
This option specifies the application name under which resources are to be obtained, rather than the default executable file name. *Name* should not contain "." or "*" characters.

–title *string*

This option specifies the window title string, which may be displayed by window managers if the user so chooses. The default title is the command line specified after the **-e** option, if any, otherwise the application 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.

RESOURCES

The program understands all of the core X Toolkit resource names and classes as well as:

iconGeometry (class **IconGeometry**)
Specifies the preferred size and position of the application when iconified. It is not necessarily obeyed by all window managers.

termName (class **TermName**)
Specifies the terminal type name to be set in the TERM environment variable.

title (class **Title**)
Specifies a string that may be used by the window manager when displaying this application.

ttyModes (class **TtyModes**)
Specifies a string containing terminal setting keywords and the characters to which they may be bound. Allowable keywords include: intr, quit, erase, kill, eof, eol, swtch, start, stop, brk, susp, dsusp, rprnt, flush, weras, and lnext. Control characters may be specified as ^char (e.g. ^c or ^u) and ^? may be used to indicate delete. This is very useful for overriding the default terminal settings without having to do an *stty* every time an *xterm* is started.

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*):

allowSendEvents (class **AllowSendEvents**)
Specifies whether or not synthetic key and button events (generated using the X protocol SendEvent request) should be interpreted or discarded. The default is "false" meaning they are discarded. Note that allowing such events creates a very large security hole.

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.

boldFont (class **Font**)
Specifies the name of the bold font to use instead of overstriking.

c132 (class **C132**)
Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is "false."

charClass (class **CharClass**)
Specifies comma-separated lists of character class bindings of the form [*low*-]*high*:*value*. These are used in determining which sets of characters should be treated the same when doing cut and paste. See the section on specifying character classes.

curses (class **Curses**)
Specifies whether or not the last column bug in *curses*(3x) 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."

eightBitInput (class **EightBitInput**)
Specifies whether or not eight-bit characters should be accepted. The default is "false."

eightBitOutput (class **EightBitOutput**)
Specifies whether eight-bit characters should be displayed. The default is "true."

font (class **Font**)
Specifies the name of the normal font. The default is "vtsingle."

font1 (class **Font1**)
Specifies the name of the first alternate font.

font2 (class **Font2**)
Specifies the name of the second alternate font.

font3 (class **Font3**)
Specifies the name of the third alternate font.

font4 (class **Font4**)
Specifies the name of the fourth alternate font.

geometry (class **Geometry**)
Specifies the preferred size and position of the VT102 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 "true".

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."

multiClickTime (class **MultiClickTime**)
Specifies the maximum time in milliseconds between multi-clock select events. The default is 250 milliseconds.

multiScroll (class **MultiScroll**)
Specifies whether or not scrolling should be done asynchronously. 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 foreground color of the pointer. The default is "XtDefaultForeground."

pointerColorBackground (class **Background**)
Specifies the background color of the pointer. The default is "XtDefaultBackground."

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."

scrollLines (class **ScrollLines**)
Specifies the number of lines that the *scroll-back* and *scroll-forw* actions should use as a default. The default value is 1.

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."

tekGeometry (class **Geometry**)
Specifies the preferred size and position of the Tektronix window.

tekInhibit (class **TekInhibit**)
Specifies whether or not Tektronix mode should be disallowed. The default is "false."

tekSmall (class **TekSmall**)
Specifies whether or not the Tektronix mode window should start in its smallest size if no explicit geometry is given. This is useful when running *xterm* on displays with small screens. The default is "false."

tekStartup (class **TekStartup**)
Specifies whether or not *xterm* should start up in Tektronix mode. The default is "false."

titeInhibit (class **TiteInhibit**)
Specifies whether or not *xterm* should remove remove *ti* or *te* termcap entries (used to switch between alternate screens on startup of many screen-oriented programs) from the TERMCAP string.

translations (class **Translations**)
Specifies the key and button bindings for menus, selections, "programmed strings", etc. See **ACTIONS** below.

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."

waitForMap (class **WaitForMap**)
Specifies whether or not *xterm* should wait for the initial window map before starting the subprocess. 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.

fontLarge (class **Font**)
Specifies the large font to use in the Tektronix window.

font2 (class **Font**)
Specifies font number 2 to use in the Tektronix window.

font3 (class **Font**)
Specifies font number 2 font to use in the Tektronix window.

fontSmall (class **Font**)
Specifies the small font to use in the Tektronix window.

The resources that may be specified for the various menus are described in the documentation for the Athena **SimpleMenu** widget. The name and classes of the entries in each of the menus are listed below.

The *mainMenu* has the following entries:

securekbd (class **SmeBSB**)
This entry invokes the **secure()** action.

allowsends (class **SmeBSB**)

This entry invokes the **allow-send-events(toggle)** action.

logging (class **SmeBSB**)
This entry invokes the **set-logging(toggle)** action.

redraw (class **SmeBSB**)
This entry invokes the **redraw()** action.

line1 (class **SmeLine**)
This is a separator.

suspend (class **SmeBSB**)
This entry invokes the **send-signal(suspend)** action on systems that support job control.

continue (class **SmeBSB**)
This entry invokes the **send-signal(cont)** action on systems that support job control.

interrupt (class **SmeBSB**)
This entry invokes the **send-signal(int)** action.

hangup (class **SmeBSB**)
This entry invokes the **send-signal(hup)** action.

terminate (class **SmeBSB**)
This entry invokes the **send-signal(term)** action.

kill (class **SmeBSB**)
This entry invokes the **send-signal(kill)** action.

line2 (class **SmeLine**)
This is a separator.

quit (class **SmeBSB**)
This entry invokes the **quit()** action.

The *vtMenu* has the following entries:

scrollbar (class **SmeBSB**)
This entry invokes the **set-scrollbar(toggle)** action.

jumpscroll (class **SmeBSB**)
This entry invokes the **set-jumpscroll(toggle)** action.

reversevideo (class **SmeBSB**)
This entry invokes the **set-reverse-video(toggle)** action.

autowrap (class **SmeBSB**)
This entry invokes the **set-autowrap(toggle)** action.

reversewrap (class **SmeBSB**)
This entry invokes the **set-reversewrap(toggle)** action.

autolinefeed (class **SmeBSB**)
This entry invokes the **set-autolinefeed(toggle)** action.

appcursor (class **SmeBSB**)
This entry invokes the **set-appcursor(toggle)** action.

appkeypad (class **SmeBSB**)
This entry invokes the **set-appkeypad(toggle)** action.

scrollkey (class **SmeBSB**)
This entry invokes the **set-scroll-on-key(toggle)** action.

scrollttyoutput (class **SmeBSB**)

This entry invokes the **set-scroll-on-tty-output(toggle)** action.

allow132 (class **SmeBSB**)
This entry invokes the **set-allow132(toggle)** action.

cursesemul (class **SmeBSB**)
This entry invokes the **set-cursesemul(toggle)** action.

visualbell (class **SmeBSB**)
This entry invokes the **set-visualbell(toggle)** action.

marginbell (class **SmeBSB**)
This entry invokes the **set-marginbell(toggle)** action.

altscreen (class **SmeBSB**)
This entry is currently disabled.

line1 (class **SmeLine**)
This is a separator.

softreset (class **SmeBSB**)
This entry invokes the **soft-reset()** action.

hardreset (class **SmeBSB**)
This entry invokes the **hard-reset()** action.

line2 (class **SmeLine**)
This is a separator.

tekshow (class **SmeBSB**)
This entry invokes the **set-visibility(tek,toggle)** action.

tekmode (class **SmeBSB**)
This entry invokes the **set-terminal-type(tek)** action.

vthide (class **SmeBSB**)
This entry invokes the **set-visibility(vt,off)** action.

The *fontMenu* has the following entries:

fontdefault (class **SmeBSB**)
This entry invokes the **set-vt-font(d)** action.

font1 (class **SmeBSB**)
This entry invokes the **set-vt-font(1)** action.

font2 (class **SmeBSB**)
This entry invokes the **set-vt-font(2)** action.

font3 (class **SmeBSB**)
This entry invokes the **set-vt-font(3)** action.

font4 (class **SmeBSB**)
This entry invokes the **set-vt-font(4)** action.

fontescape (class **SmeBSB**)
This entry invokes the **set-vt-font(e)** action.

fontsel (class **SmeBSB**)
This entry invokes the **set-vt-font(s)** action.

The *tekMenu* has the following entries:

tektextlarge (class **SmeBSB**)
This entry invokes the **set-tek-text(l)** action.

tektext2 (class **SmeBSB**)
This entry invokes the **set-tek-text(2)** action.

tektext3 (class **SmeBSB**)
This entry invokes the **set-tek-text(3)** action.

tektextsmall (class **SmeBSB**)
This entry invokes the **set-tek-text(s)** action.

line1 (class **SmeLine**)
This is a separator.

tekpage (class **SmeBSB**)
This entry invokes the **tek-page()** action.

tekreset (class **SmeBSB**)
This entry invokes the **tek-reset()** action.

tekcopy (class **SmeBSB**)
This entry invokes the **tek-copy()** action.

line2 (class **SmeLine**)
This is a separator.

vtshow (class **SmeBSB**)
This entry invokes the **set-visibility(vt,toggle)** action.

vtmode (class **SmeBSB**)
This entry invokes the **set-terminal-type(vt)** action.

tekhide (class **SmeBSB**)
This entry invokes the **set-visibility(tek,toggle)** action.

The following resources are useful when specified for the Athena Scrollbar widget:

thickness (class **Thickness**)
Specifies the width in pixels of the scrollbar.

background (class **Background**)
Specifies the color to use for the background of the scrollbar.

foreground (class **Foreground**)
Specifies the color to use for the foreground of the scrollbar. The "thumb" of the scrollbar is a simple checkerboard pattern alternating pixels for foreground and background color.

EMULATIONS

The VT102 emulation is fairly complete, but does not support the blinking character attribute nor the double-wide and double-size character sets. *Termcap*(5) entries that work with *xterm* include "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences. (See the *"Xterm Control Sequences"* document.)

The Tektronix 4014 emulation is also fairly good. Four different font sizes and five different lines types are supported. The Tektronix text and graphics commands are recorded internally by *xterm* and may be written to a file by sending the COPY escape sequence (or through the **Tektronix** menu; see below). The name of the file will be "**COPY***yy–MM–dd.hh:mm:ss*", where *yy*, *MM*, *dd*, *hh*, *mm* and *ss* are the year, month, day, hour, minute and second when the COPY was performed (the file is created in the directory *xterm* is started in, or the home directory for a login

xterm).

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key. The assignment of the functions described below to keys and buttons may be changed through the resource database; see **ACTIONS** below.

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 and made the PRIMARY selection 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. If the key/button bindings specify that an X selection is to be made, *xterm* will leave the selected text highlighted for as long as it is the selection owner.

Pointer button two (usually middle) 'types' (pastes) the text from the PRIMARY selection, if any, otherwise 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 dows 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 is bit is normally stripped unless the terminal mode is RAW; see *tty*(4) for details).

MENUS

Xterm has four menus, named *mainMenu*, *vtMenu*, *fontMenu*, and *tekMenu*. Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two section, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. The *mainMenu* contains items that apply to both the VT102 and Tektronix windows. The **Secure Keyboard** mode is be used when typing in passwords or other sensitive data in an unsecure environment; see **SECURITY** below. 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 *vtMenu* 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 *fontMenu* sets the font used in the VT102 window.

The *tekMenu* 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.

SECURITY

X environments differ in their security consciousness. MIT servers, run under *xdm*, are capable of using a "magic cookie" authorization scheme that can provide a reasonable level of security for many people. If your server is only using a host-based mechanism to control access to the server (see *xhost(1)*), then if you enable access for a host and other users are also permitted to run clients on that same host, there is every possibility that someone can run an application that will use the basic services of the X protocol to snoop on your activities, potentially capturing a transcript of everything you type at the keyboard. This is of particular concern when you want to type in a password or other sensitive data. The best solution to this problem is to use a better authorization mechanism that host-based control, but a simple mechanism exists for protecting keyboard input in *xterm*.

The **xterm** menu (see **MENUS** above) contains a **Secure Keyboard** entry which, when enabled, ensures that all keyboard input is directed *only* to *xterm* (using the GrabKeyboard protocol request). When an application prompts you for a password (or other sensitive data), you can enable **Secure Keyboard** using the menu, type in the data, and then disable **Secure Keyboard** using the menu again. Only one X client at a time can secure the keyboard, so when you attempt to enable **Secure Keyboard** it may fail. In this case, the bell will sound. If the **Secure Keyboard** succeeds, the foreground and background colors will be exchanged (as if you selected the **Reverse Video** entry in the **Modes** menu); they will be exchanged again when you exit secure mode. If the colors do *not* switch, then you should be *very* suspicious that you are being spoofed. If the application you are running displays a prompt before asking for the password, it is safest to enter secure mode *before* the prompt gets displayed, and to make sure that the prompt gets displayed correctly (in the new colors), to minimize the probability of spoofing. You can also bring up the menu again and make sure that a check mark appears next to the entry.

Secure Keyboard mode will be disabled automatically if your xterm window becomes iconified (or otherwise unmapped), or if you start up a reparenting window manager (that places a title bar or other decoration around the window) while in **Secure Keyboard** mode. (This is a feature of the X protocol not easily overcome.) When this happens, the foreground and background colors will be switched back and the bell will sound in warning.

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 *charClass* (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 filenames.

ACTIONS

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 and button events is not expected, and will cause unpredictable behavior. The following actions are provided for using within the *vt100* or *tek4014* **translations** resources:

bell([*percent*]**)**
This action rings the keyboard bell at the specified percentage above or below the base volume.

ignore()
This action ignores the event but checks for special pointer position escape sequences.

insert() This action is a synonym for **insert-seven-bit()**

insert-seven-bit()
This action inserts the 7-bit USASCII character or string associated with the keysym that was pressed.

insert-eight-bit()
This action inserts the 8-bit ISO Latin-1 character or string associated with the keysym that was pressed.

insert-selection(*sourcename* [, ...]**)**
This action inserts the string found in the selection or cutbuffer indicated by *sourcename*. Sources are checked in the order given (case is significant) until one is found. Commonly-used selections include: *PRIMARY*, *SECONDARY*, and *CLIPBOARD*. Cut buffers are typically named *CUT_BUFFER0* through *CUT_BUFFER7*.

keymap(*name***)**
This action dynamically defines a new translation table whose resource name is *name* with the suffix *Keymap* (case is significant). The name *None* restores the original translation table.

popup-menu(*menuname***)**
This action displays the specified popup menu. Valid names (case is significant) include: *mainMenu*, *vtMenu*, *fontMenu*, and *tekMenu*.

secure()
This action toggles the *Secure Keyboard* mode described in the section named **SECURITY**, and is invoked from the **securekbd** entry in *mainMenu*.

select-start()
This action begins text selection at the current pointer location. See the section on **POINTER USAGE** for information on making selections.

select-extend()
This action tracks the pointer and extends the selection. It should only be bound to Motion events.

select-end(*destname* [, ...]**)**
This action puts the currently selected text into all of the selections or cutbuffers specified by *destname*.

select-cursor-start()
This action is similar to **select-start** except that it begins the selection at the current text cursor position.

select-cursor-end(*destname* [, ...])
This action is similar to **select-end** except that it should be used with **select-cursor-start.**

set-vt-font(*d/1/2/3/4/e/s* [,*normalfont* [, *boldfont*]])
This action sets the font or fonts currently being used in the VT102 window. The first argument is a single character that specifies the font to be used: *d* or *D* indicate the default font (the font initially used when *xterm* was started), *1* through *4* indicate the fonts specified by the *font1* through *font4* resources, *e* or *E* indicate the normal and bold fonts that may be set through escape codes (or specified as the second and third action arguments, respectively), and *i* or *I* indicate the font selection (as made by programs such as *xfontsel(1)*) indicated by the second action argument.

start-extend()
This action is similar to **select-start except that the selection is extended to the current pointer location.**

start-cursor-extend()
This action is similar to **select-extend** except that the selection is extended to the current text cursor position.

string(*string*)
This action inserts the specified text string as if it had been typed. Quotation is necessary if the string contains whitespace or non-alphanumeric characters. If the string argument begins with the characters "0x", it is interpreted as a hex character constant.

scroll-back(*count* [,*units*])
This action scrolls the text window backward so that text that had previously scrolled off the top of the screen is now visible. The *count* argument indicates the number of *units* (which may be *page*, *halfpage*, *pixel*, or *line*) by which to scroll.

scroll-forw(*count* [,*units*])
This action scrolls is similar to **scroll-back** except that it scrolls the other direction.

allow-send-events(*on/off/toggle*)
This action set or toggles the **allowSendEvents** resource and is also invoked by the **allowsends** entry in *mainMenu.*

set-logging(*on/off/toggle*)
This action toggles the **logging** resource and is also invoked by the **logging** entry in *mainMenu.*

redraw()
This action redraws the window and is also invoked by the *redraw* entry in *mainMenu.*

send-signal(*signame*)
This action sends the signal named by *signame* (which may also be a number) to the *xterm* subprocess (the shell or program specified with the *-e* command line option) and is also invoked by the **suspend**, **continue**, **interrupt**, **hangup**, **terminate**, and *kill* entries in *mainMenu.* Allowable signal names are (case is not significant): *suspend*, *tstp* (if supported by the operating system), *cont* (if supported by the operating system), *int*, *hup*, *term*, and *kill.*

quit() This action sends a SIGHUP to the subprogram and exits. It is also invoked by the **quit** entry in *mainMenu.*

set-scrollbar(*on/off/toggle*)
This action toggles the **scrollbar** resource and is also invoked by the **scrollbar** entry in *vtMenu.*

set-jumpscroll(*on/off/toggle*)

This action toggles the **jumpscroll** resource and is also invoked by the **jumpscroll** entry in *vtMenu*.

set-reverse-video(*on/off/toggle***)**
This action toggles the *reverseVideo* resource and is also invoked by the **reversevideo** entry in *vtMenu*.

set-autowrap(*on/off/toggle***)**
This action toggles automatic wrapping of long lines and is also invoked by the **autowrap** entry in *vtMenu*.

set-reversewrap(*on/off/toggle***)**
This action toggles the **reverseWrap** resource and is also invoked by the **reversewrap** entry in *vtMenu*.

set-autolinefeed(*on/off/toggle***)**
This action toggles automatic insertion of linefeeds and is also invoked by the **autolinefeed** entry in *vtMenu*.

set-appcursor(*on/off/toggle***)**
This action toggles the handling Application Cursor Key mode and is also invoked by the Bappcursor entry in *vtMenu*.

set-appkeypad(*on/off/toggle***)**
This action toggles the handling of Application Keypad mode and is also invoked by the **appkeypad** entry in *vtMenu*.

set-scroll-on-key(*on/off/toggle***)**
This action toggles the **scrollKey** resource and is also invoked from the **scrollkey** entry in *vtMenu*.

set-scroll-on-tty-output(*on/off/toggle***)**
This action toggles the **scrollTtyOutput** resource and is also invoked from the **scrolltyoutput** entry in *vtMenu*.

set-allow132(*on/off/toggle***)**
This action toggles the **c132** resource and is also invoked from the **allow132** entry in *vtMenu*.

set-cursesemul(*on/off/toggle***)**
This action toggles the **curses** resource and is also invoked from the **cursesemul** entry in *vtMenu*.

set-visual-bell(*on/off/toggle***)**
This action toggles the **visualBell** resource and is also invoked by the **visualbell** entry in *vtMenu*.

set-marginbell(*on/off/toggle***)**
This action toggles the **marginBell** resource and is also invoked from the **marginbell** entry in *vtMenu*.

set-altscreen(*on/off/toggle***)**
This action toggles between the alternative and current screens.

soft-reset()
This action resets the scrolling region and is also invoked from the **softreset** entry in *vtMenu*.

hard-reset()
This action resets the scrolling region, tabs, window size, and cursor keys and clears the screen. It is also invoked from the **hardreset** entry in *vtMenu*.

set-terminal-type(*type***)**

This action directs output to either the *vt* or *tek* windows, according to the *type* string. It is also invoked by the **tekmode** entry in *vtMenu* and the **vtmode** entry in *tekMenu*.

set-visibility(*vt/tek,on/off/toggle*)
This action controls whether or not the *vt* or *tek* windows are visible. It is also invoked from the **tekshow** and **vthide** entries in *vtMenu* and the **vtshow** and **tekhide** entries in *tekMenu*.

set-tek-text(*large/2/3/small*)
This action sets font used in the Tektronix window to the value of the resources **tektextlarge**, **tektext2**, **tektext3**, and **tektextsmall** according to the argument. It is also by the entries of the same names as the resources in *tekMenu*.

tek-page()
This action clears the Tektronix window and is also invoked by the **tekpage** entry in *tekMenu*.

tek-reset()
This action resets the Tektronix window and is also invoked by the *tekreset* entry in *tekMenu*.

tek-copy()
This action copies the escape codes used to generate the current window contents to a file in the current directory beginning with the name COPY. It is also invoked from the *tekcopy* entry in *tekMenu*.

The Tektronix window also has the following action:

gin-press(*l/L/m/M/r/R*)
This action send the indicated graphics input code.

The default bindings in the VT102 window are:

```
 Shift <KeyPress> Prior:          scroll-back(1,halfpage) \n\
  Shift <KeyPress> Next:          scroll-forw(1,halfpage) \n\
Shift <KeyPress> Select:          select-cursor-start() \
                                  select-cursor-end(PRIMARY, CUT_BUFFER0) \n\
Shift <KeyPress> Insert:          insert-selection(PRIMARY, CUT_BUFFER0) \n\
      ~Meta<KeyPress>:            insert-seven-bit() \n\
       Meta<KeyPress>:            insert-eight-bit() \n\
  Ctrl ~Meta<Btn1Down>:           popup-menu(mainMenu) \n\
  Ctrl Shift<Btn1Down>:           popup-menu(mainMenu) \n\
     ~Meta <Btn1Down>:            select-start() \n\
    ~Meta <Btn1Motion>:           select-extend() \n\
 Ctrl ~Meta <Btn2Down>:           popup-menu(vtMenu) \n\
 Ctrl Shift <Btn2Down>:           popup-menu(vtMenu) \n\
~Ctrl ~Meta <Btn2Down>:           ignore() \n\
 ~Ctrl ~Meta <Btn2Up>:            insert-selection(PRIMARY, CUT_BUFFER0) \n\
 Ctrl ~Meta <Btn3Down>:           popup-menu(fontMenu) \n\
 Ctrl Shift <Btn3Down>:           popup-menu(fontMenu) \n\
~Ctrl ~Meta <Btn3Down>:           start-extend() \n\
    ~Meta <Btn3Motion>:           select-extend() \n\
  ~Ctrl ~Meta <BtnUp>:            select-end(PRIMARY, CUT_BUFFER0) \n\
        <BtnDown>:                bell(0)
```

The default bindings in the Tektronix window are:

```
~Meta<KeyPress>:                  insert-seven-bit() \n\
```

```
    Meta<KeyPress>:                 insert-eight-bit()\n\
 Ctrl ~Meta<Btn1Down>:              popup-menu(mainMenu) \n\
Ctrl ~Meta <Btn2Down>:              popup-menu(tekMenu) \n\
Shift ~Meta<Btn1Down>:              gin-press(L) \n\
    ~Meta<Btn1Down>:                gin-press(l) \n\
Shift ~Meta<Btn2Down>:              gin-press(M) \n\
    ~Meta<Btn2Down>:                gin-press(m) \n\
Shift ~Meta<Btn3Down>:              gin-press(R) \n\
    ~Meta<Btn3Down>:                gin-press(r)
```

Below is a sample how of the **keymap()** action is used to add special keys for entering commonly-typed works:

```
*VT100.Translations: #override <Key>F13: keymap(dbx)
*VT100.dbxKeymap.translations: \
        <Key>F14:    keymap(None) \n\
        <Key>F17:    string("next") string(0x0d) \n\
        <Key>F18:    string("step") string(0x0d) \n\
        <Key>F19:    string("continue") string(0x0d) \n\
        <Key>F20:    string("print ") insert-selection(PRIMARY, CUT_BUFFER0)
```

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replace with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The *termcap*(5) entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

In either VT102 or Tektronix mode, there are escape sequences to change the name of the windows and to specify a new log file name.

ENVIRONMENT

Xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

SEE ALSO

resize(1), X(1), pty(4), tty(4)
Xterm Control Sequences

BUGS

The *Xterm Control Sequences* document has yet to be converted from X10. The old version, along with a first stab at an update, are available in the sources.

The class name is *XTerm* instead of *Xterm*.

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.

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.

The Tek widget does not support key/button re-binding.

COPYRIGHT

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), Dave Serisky (HP)

NOTE

xterm is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xtraining - introduction to CXwindows, Motif, and mwm

SYNOPSIS

xtraining [*-toolkitoption ...*]

DESCRIPTION

xtraining is an on-line introduction to the CXwindows environment. It explains the basic concepts and terms related to using X windows and how to use the the Motif Window Manager.

In order for the tutorial to be consistent with what you see, you must have CXwindows V2.0 installed and running before you invoke *xtraining.*

You can invoke *xtraining* with no window manager running, but you must have a .mwmrc file in your home directory. You get it by entering

```
% cp /usr/lib/X11/system.mwmrc ~/.mwmrc
```

from a csh prompt. You may also run *xtraining* with mwm already running.

If you are running another window manager, you will not see everything *xtraining* describes.

Be sure that the directory containing the X clients (/usr/bin/X11) is in your search path ($PATH).

OPTIONS

xtraining accepts all of the standard X Toolkit command line options.

SEE ALSO

X(1), mwm(1)

COPYRIGHT

NOTES

xtraining is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

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 'Usage:' 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

AUTHORS

Tony Della Fera, Digital Equipment Corp., MIT Project Athena
William F. Wyatt, Smithsonian Astrophysical Observatory

NOTE

xwd is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xwininfo - window information utility for X

SYNOPSIS

xwininfo [-help] [-id *id*] [-root] [-name *name*] [-int] [-tree] [-stats] [-bits] [-events] [-size] [-wm] [-frame] [-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.

-frame This option causes window manager frames not be ignored when manually selecting windows.

-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.

AUTHOR

Mark Lillibridge, MIT Project Athena

NOTE

xwininfo is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

xwud - image displayer for X

SYNOPSIS

xwud [-in *file*] [-noclick] [-geometry *geom*] [-display *display*] [-new] [-std <maptype>] [-raw] [-vis <vis-type-or-id>] [-help] [-rv] [-plane *number*] [-fg *color*] [-bg *color*]

DESCRIPTION

Xwud is an X Window System image undumping utility. *Xwud* allows X users to display in a window an image saved in a specially formatted dump file, such as produced by *xwd(1)*.

OPTIONS

-bg *color*

If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "0" bits in the image.

-display *display*

This option allows you to specify the server to connect to; see *X(1)*.

-fg *color*

If a bitmap image (or a single plane of an image) is displayed, this option can be used to specify the color to display for the "1" bits in the image.

-geometry *geom*

This option allows you to specify the size and position of the window. Typically you will only want to specify the position, and let the size default to the actual size of the image.

-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.

-new This option forces creation of a new colormap for displaying the image. If the image characteristics happen to match those of the display, this can get the image on the screen faster, but at the cost of using a new colormap (which on most displays will cause other windows to go technicolor).

-noclick

Clicking any button in the window will terminate the application, unless this option is specified. Termination can always be achieved by typing 'q', 'Q', or ctrl-c.

-plane *number*

You can select a single bit plane of the image to display with this option. Planes are numbered with zero being the least significant bit. This option can be used to figure out which plane to pass to *xpr(1)* for printing.

-raw This option forces the image to be displayed with whatever color values happen to currently exist on the screen. This option is mostly useful when undumping an image back onto the same screen that the image originally came from, while the original windows are still on the screen, and results in getting the image on the screen faster.

-rv If a bitmap image (or a single plane of an image) is displayed, this option forces the foreground and background colors to be swapped. This may be needed when displaying a bitmap image which has the color sense of pixel values "0" and "1" reversed from what they are on your display.

-std *maptype*

This option causes the image to be displayed using the specified Standard Colormap. The property name is obtained by converting the type to upper case, prepending "RGB_", and appending "_MAP". Typical types are "best", "default", and "gray". See *xcmap(1)* for one way of creating Standard Colormaps.

-vis *vis-type-or-id*

This option allows you to specify a particular visual or visual class. The default is to pick the "best" one. A particular class can be specified: "StaticGray", "GrayScale", "StaticColor", "PseudoColor", "DirectColor", or "TrueColor". Or "Match" can be specified, meaning use the same class as the source image. Alternatively, an exact visual id (specific to the server) can be specified, either as a hexadecimal number (prefixed with "0x") or as a decimal number. Finally, "default" can be specified, meaning to use the same class as the colormap of the root window. Case is not significant in any of these strings.

ENVIRONMENT

DISPLAY

To get default display.

FILES

XWDFile.h

X Window Dump File format definition file.

SEE ALSO

xwd(1), xpr(1), xcmap(1), X(1)

COPYRIGHT

AUTHOR

Bob Scheifler, MIT X Consortium

NOTE

xwud is part of CXwindows, an optional product; for more information contact your CONVEX sales representative.

NAME

AllPlanes, BlackPixel, WhitePixel, ConnectionNumber, DefaultColormap, DefaultDepth, XListDepths, DefaultGC, DefaultRootWindow, DefaultScreenOfDisplay, DefaultScreen, DefaultVisual, DisplayCells, DisplayPlanes, DisplayString, LastKnownRequestProcessed, NextRequest, ProtocolVersion, ProtocolRevision, QLength, RootWindow, ScreenCount, ScreenOfDisplay, ServerVendor, VendorRelease – Display macros

SYNTAX

AllPlanes

BlackPixel(*display, screen_number*)

WhitePixel(*display, screen_number*)

ConnectionNumber(*display*)

DefaultColormap(*display, screen_number*)

DefaultDepth(*display, screen_number*)

int *XListDepths(*display, screen_number, count_return*)
Display **display*;
int *screen_number*;
int **count_return*;

DefaultGC(*display, screen_number*)

DefaultRootWindow(*display*)

DefaultScreenOfDisplay(*display*)

DefaultScreen(*display*)

DefaultVisual(*display, screen_number*)

DisplayCells(*display, screen_number*)

DisplayPlanes(*display, screen_number*)

DisplayString(*display*)

LastKnownRequestProcessed(*display*)

NextRequest(*display*)

ProtocolVersion(*display*)

ProtocolRevision(*display*)

QLength(*display*)

RootWindow(*display, screen_number*)

ScreenCount(*display*)

ScreenOfDisplay(*display, screen_number*)

ServerVendor(*display*)

VendorRelease(*display*)

ARGUMENTS

display	Specifies the connection to the X server.
screen_number	Specifies the appropriate screen number on the host server.
count_return	Returns the number of depths.

DESCRIPTION

The **AllPlanes** macro returns a value with all bits set to 1 suitable for use in a plane argument to a procedure.

The **BlackPixel** macro returns the black pixel value for the specified screen.

The **WhitePixel** macro returns the white pixel value for the specified screen.

The **ConnectionNumber** macro returns a connection number for the specified display.

The **DefaultColormap** macro returns the default colormap ID for allocation on the specified screen.

The **DefaultDepth** macro returns the depth (number of planes) of the default root window for the specified screen.

The **XListDepths** function returns the array of depths that are available on the specified screen. If the specified screen_number is valid and sufficient memory for the array can be allocated, **XListDepths** sets count_return to the number of available depths. Otherwise, it does not set count_return and returns NULL. To release the memory allocated for the array of depths, use **XFree**.

The **DefaultGC** macro returns the default GC for the root window of the specified screen.

The **DefaultRootWindow** macro returns the root window for the default screen.

The **DefaultScreenOfDisplay** macro returns the default screen of the specified display.

The **DefaultScreen** macro returns the default screen number referenced in the **XOpenDisplay** routine.

The **DefaultVisual** macro returns the default visual type for the specified screen.

The **DisplayCells** macro returns the number of entries in the default colormap.

The **DisplayPlanes** macro returns the depth of the root window of the specified screen.

The **DisplayString** macro returns the string that was passed to **XOpenDisplay** when the current display was opened.

The **LastKnownRequestProcessed** macro extracts the full serial number of the last request known by Xlib to have been processed by the X server.

The **NextRequest** macro extracts the full serial number that is to be used for the next request.

The **ProtocolVersion** macro returns the major version number (11) of the X protocol associated with the connected display.

The **ProtocolRevision** macro returns the minor protocol revision number of the X server.

The **QLength** macro returns the length of the event queue for the connected display.

The **RootWindow** macro returns the root window.

The **ScreenCount** macro returns the number of available screens.

The **ScreenOfDisplay** macro returns a pointer to the screen of the specified display.

The **ServerVendor** macro returns a pointer to a null-terminated string that provides some identification of the owner of the X server implementation.

The **VendorRelease** macro returns a number related to a vendor's release of the X server.

SEE ALSO

BlackPixelOfScreen(3X11), ImageByteOrder(3X11), IsCursorKey(3X11), XOpenDisplay(3X11)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

BlackPixelOfScreen, WhitePixelOfScreen, CellsOfScreen, DefaultColormapOfScreen, DefaultDepthOfScreen, DefaultGCOfScreen, DefaultVisualOfScreen, DoesBackingStore, DoesSaveUnders, DisplayOfScreen, XScreenNumberOfScreen, EventMaskOfScreen, HeightOfScreen, HeightMMOfScreen, MaxCmapsOfScreen, MinCmapsOfScreen, PlanesOfScreen, RootWindowOfScreen, WidthOfScreen, WidthMMOfScreen - screen information functions and macros

SYNTAX

BlackPixelOfScreen(*screen*)

WhitePixelOfScreen(*screen*)

CellsOfScreen(*screen*)

DefaultColormapOfScreen(*screen*)

DefaultDepthOfScreen(*screen*)

DefaultGCOfScreen(*screen*)

DefaultVisualOfScreen(*screen*)

DoesBackingStore(*screen*)

DoesSaveUnders(*screen*)

DisplayOfScreen(*screen*)

int XScreenNumberOfScreen(*screen*)
    Screen **screen*;

EventMaskOfScreen(*screen*)

HeightOfScreen(*screen*)

HeightMMOfScreen(*screen*)

MaxCmapsOfScreen(*screen*)

MinCmapsOfScreen(*screen*)

PlanesOfScreen(*screen*)

RootWindowOfScreen(*screen*)

WidthOfScreen(*screen*)

WidthMMOfScreen(*screen*)

ARGUMENTS

screen Specifies a pointer to the appropriate **Screen** structure.

DESCRIPTION

The **BlackPixelOfScreen** macro returns the black pixel value of the specified screen.

The **WhitePixelOfScreen** macro returns the white pixel value of the specified screen.

The **CellsOfScreen** macro returns the number of colormap cells in the default colormap of the specified screen.

The **DefaultColormapOfScreen** macro returns the default colormap of the specified screen.

The **DefaultDepthOfScreen** macro returns the default depth of the root window of the specified screen.

The **DefaultGCOfScreen** macro returns the default GC of the specified screen, which has the same depth as the root window of the screen.

The **DefaultVisualOfScreen** macro returns the default visual of the specified screen.

The **DoesBackingStore** macro returns **WhenMapped**, **NotUseful**, or **Always**, which indicate whether the screen supports backing stores.

The **DoesSaveUnders** macro returns a Boolean value indicating whether the screen supports save unders.

The **DisplayOfScreen** macro returns the display of the specified screen.

The **XScreenNumberOfScreen** function returns the screen index number of the specified screen.

The **EventMaskOfScreen** macro returns the root event mask of the root window for the specified screen at connection setup.

The **HeightOfScreen** macro returns the height of the specified screen.

The **HeightMMOfScreen** macro returns the height of the specified screen in millimeters.

The **MaxCmapsOfScreen** macro returns the maximum number of installed colormaps supported by the specified screen.

The **MinCmapsOfScreen** macro returns the minimum number of installed colormaps supported by the specified screen.

The **PlanesOfScreen** macro returns the number of planes in the root window of the specified screen.

The **RootWindowOfScreen** macro returns the root window of the specified screen.

The **WidthOfScreen** macro returns the width of the specified screen.

The **WidthMMOfScreen** macro returns the width of the specified screen in millimeters.

SEE ALSO

AllPlanes(3X11), ImageByteOrder(3X11), IsCursorKey(3X11)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

ImageByteOrder, BitmapBitOrder, BitmapPad, BitmapUnit, DisplayHeight, DisplayHeightMM, DisplayWidth, DisplayWidthMM, XListPixmapFormats, XPixmapFormatValues – image format functions and macros

SYNTAX

XPixmapFormatValues *XListPixmapFormats(*display, count_return*)
Display **display*;
int **count_return*;

ImageByteOrder(*display*)

BitmapBitOrder(*display*)

BitmapPad(*display*)

BitmapUnit(*display*)

DisplayHeight(*display, screen_number*)

DisplayHeightMM(*display, screen_number*)

DisplayWidth(*display, screen_number*)

DisplayWidthMM(*display, screen_number*)

ARGUMENTS

display Specifies the connection to the X server.

count_return Returns the number of pixmap formats that are supported by the display.

screen_number Specifies the appropriate screen number on the host server.

DESCRIPTION

The **XListPixmapFormats** function returns an array of **XPixmapFormatValues** structures that describe the types of Z format images that are supported by the specified display. If insufficient memory is available, **XListPixmapFormats** returns NULL. To free the allocated storage for the **XPixmapFormatValues** structures, use **XFree**.

The **ImageByteOrder** macro specifies the required byte order for images for each scanline unit in XY format (bitmap) or for each pixel value in Z format.

The **BitmapBitOrder** macro returns **LSBFirst** or **MSBFirst** to indicate whether the leftmost bit in the bitmap as displayed on the screen is the least or most significant bit in the unit.

The **BitmapPad** macro returns the number of bits that each scanline must be padded.

The **BitmapUnit** macro returns the size of a bitmap's scanline unit in bits.

The **DisplayHeight** macro returns the height of the specified screen in pixels.

The **DisplayHeightMM** macro returns the height of the specified screen in millimeters.

The **DisplayWidth** macro returns the width of the screen in pixels.

The **DisplayWidthMM** macro returns the width of the specified screen in millimeters.

STRUCTURES

The **XPixmapFormatValues** structure provides an interface to the pixmap format information that is returned at the time of a connection setup. It contains:

```
typedef struct {
        int depth;
        int bits_per_pixel;
        int scanline_pad;
} XPixmapFormatValues;
```

SEE ALSO

AllPlanes(3X11), BlackPixelOfScreen(3X11), IsCursorKey(3X11), XFree(3X11)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

IsCursorKey, IsFunctionKey, IsKeypadKey, IsMiscFunctionKey, IsModiferKey, IsPFKey – keysym classification macros

SYNTAX

IsCursorKey(*keysym*)

IsFunctionKey(*keysym*)

IsKeypadKey(*keysym*)

IsMiscFunctionKey(*keysym*)

IsModifierKey(*keysym*)

IsPFKey(*keysym*)

ARGUMENTS

keysym Specifies the KeySym that is to be tested.

DESCRIPTION

The **IsCursorKey** macro returns **True** if the specified KeySym is a cursor key.

The **IsFunctionKey** macro returns **True** if the KeySym is a function key.

The **IsKeypadKey** macro returns **True** if the specified KeySym is a keypad key.

The **IsMiscFunctionKey** macro returns **True** if the specified KeySym is a miscellaneous function key.

The **IsModiferKey** macro returns **True** if the specified KeySym is a modifier key.

The **IsPFKey** macro returns **True** if the specified KeySym is a PF key.

SEE ALSO

AllPlanes(3X11), BlackPixelOfScreen(3X11), ImageByteOrder(3X11)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAddHost, XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XSetAccessControl, XEnableAccessControl, XDisableAccessControl, XHostAddress - control host access and host control structure

SYNTAX

XAddHost(*display, host*)
Display **display*;
XHostAddress **host*;

XAddHosts(*display, hosts, num_hosts*)
Display **display*;
XHostAddress **hosts*;
int *num_hosts*;

XHostAddress *XListHosts(*display, nhosts_return, state_return*)
Display **display*;
int **nhosts_return*;
Bool **state_return*;

XRemoveHost(*display, host*)
Display **display*;
XHostAddress **host*;

XRemoveHosts(*display, hosts, num_hosts*)
Display **display*;
XHostAddress **hosts*;
int *num_hosts*;

XSetAccessControl(*display, mode*)
Display **display*;
int *mode*;

XEnableAccessControl(*display*)
Display **display*;

XDisableAccessControl(*display*)
Display **display*;

ARGUMENTS

display	Specifies the connection to the X server.
host	Specifies the host that is to be added or removed.
hosts	Specifies each host that is to be added or removed.
mode	Specifies the mode. You can pass **EnableAccess** or **DisableAccess**.
nhosts_return	Returns the number of hosts currently in the access control list.
num_hosts	Specifies the number of hosts.
state_return	Returns the state of the access control.

DESCRIPTION

The **XAddHost** function adds the specified host to the access control list for that display. The server must be on the same host as the client issuing the command, or a **BadAccess** error results.

XAddHost can generate **BadAccess** and **BadValue** errors.

The **XAddHosts** function adds each specified host to the access control list for that display. The server must be on the same host as the client issuing the command, or a **BadAccess** error results.

XAddHosts can generate **BadAccess** and **BadValue** errors.

The **XListHosts** function returns the current access control list as well as whether the use of the list at connection setup was enabled or disabled. **XListHosts** allows a program to find out what machines can make connections. It also returns a pointer to a list of host structures that were allocated by the function. When no longer needed, this memory should be freed by calling **XFree**.

The **XRemoveHost** function removes the specified host from the access control list for that display. The server must be on the same host as the client process, or a **BadAccess** error results. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed unless you reset the server.

XRemoveHost can generate **BadAccess** and **BadValue** errors.

The **XRemoveHosts** function removes each specified host from the access control list for that display. The X server must be on the same host as the client process, or a **BadAccess** error results. If you remove your machine from the access list, you can no longer connect to that server, and this operation cannot be reversed unless you reset the server.

XRemoveHosts can generate **BadAccess** and **BadValue** errors.

The **XSetAccessControl** function either enables or disables the use of the access control list at each connection setup.

XSetAccessControl can generate **BadAccess** and **BadValue** errors.

The **XEnableAccessControl** function enables the use of the access control list at each connection setup.

XEnableAccessControl can generate a **BadAccess** error.

The **XDisableAccessControl** function disables the use of the access control list at each connection setup.

XDisableAccessControl can generate a **BadAccess** error.

STRUCTURES

The **XHostAddress** structure contains:

```
typedef struct {
        int family;             /* for example FamilyInternet */
        int length;             /* length of address, in bytes */
        char *address;          /* pointer to where to find the address */
} XHostAddress;
```

The family member specifies which protocol address family to use (for example, TCP/IP or DECnet) and can be **FamilyInternet**, **FamilyDECnet**, or **FamilyChaos**. The length member specifies the length of the address in bytes. The address member specifies a pointer to the address.

DIAGNOSTICS

BadAccess A client attempted to modify the access control list from other than the local (or otherwise authorized) host.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XFree(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllocClassHint, XSetClassHint, XGetClassHint, XClassHint – allocate class hints structure and set or read a window's WM_CLASS property

SYNTAX

XClassHint *XAllocClassHint()

XSetClassHint(*display, w, class_hints*)
Display **display*;
Window *w*;
XClassHint **class_hints*;

Status XGetClassHint(*display, w, class_hints_return*)
Display **display*;
Window *w*;
XClassHint **class_hints_return*;

ARGUMENTS

display Specifies the connection to the X server.

class_hints Specifies the **XClassHint** structure that is to be used.

class_hints_return
Returns the **XClassHint** structure.

w Specifies the window.

DESCRIPTION

The **XAllocClassHint** function allocates and returns a pointer to a **XClassHint** structure. Note that the pointer fields in the **XClassHint** structure are initially set to NULL. If insufficient memory is available, **XAllocClassHint** returns NULL. To free the memory allocated to this structure, use **XFree**.

The **XSetClassHint** function sets the class hint for the specified window.

XSetClassHint can generate **BadAlloc** and **BadWindow** errors.

The **XGetClassHint** function returns the class of the specified window. To free res_name and res_class when finished with the strings, use **XFree**.

XGetClassHint can generate a **BadWindow** error.

PROPERTIES

WM_CLASS Set by application programs to allow window and session managers to obtain the application's resources from the resource database.

STRUCTURES

The **XClassHint** structure contains:

```
typedef struct {
        char *res_name;
        char *res_class;
} XClassHint;
```

The res_name member contains the application name, and the res_class member contains the application class. Note that the name set in this property may differ from the name set as WM_NAME. That is, WM_NAME specifies what should be displayed in the title bar and, therefore, can contain temporal information (for example, the name of a file currently in an editor's buffer). On the other hand, the name specified as part of WM_CLASS is the formal name of the application that should be used when retrieving the application's resources from the resource database.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllocColor, XAllocNamedColor, XAllocColorCells, XAllocColorPlanes, XFreeColors – allocate and free colors

SYNTAX

Status XAllocColor(*display, colormap, screen_in_out*)
Display **display*;
Colormap *colormap*;
XColor **screen_in_out*;

Status XAllocNamedColor(*display, colormap, color_name, screen_def_return, exact_def_return*)
Display **display*;
Colormap *colormap*;
char **color_name*;
XColor **screen_def_return, *exact_def_return*;

Status XAllocColorCells(*display, colormap, contig, plane_masks_return, nplanes, pixels_return, npixels*)
Display **display*;
Colormap *colormap*;
Bool *contig*;
unsigned long *plane_masks_return*[];
unsigned int *nplanes*;
unsigned long *pixels_return*[];
unsigned int *npixels*;

Status XAllocColorPlanes(*display, colormap, contig, pixels_return, ncolors, nreds, ngreens, nblues, rmask_return, gmask_return, bmask_return*)
Display **display*;
Colormap *colormap*;
Bool *contig*;
unsigned long *pixels_return*[];
int *ncolors*;
int *nreds, ngreens, nblues*;
unsigned long **rmask_return, *gmask_return, *bmask_return*;

XFreeColors(*display, colormap, pixels, npixels, planes*)
Display **display*;
Colormap *colormap*;
unsigned long *pixels*[];
int *npixels*;
unsigned long *planes*;

ARGUMENTS

color_name Specifies the color name string (for example, red) whose color definition structure you want returned.

colormap Specifies the colormap.

contig Specifies a Boolean value that indicates whether the planes must be contiguous.

display Specifies the connection to the X server.

exact_def_return Returns the exact RGB values.

ncolors Specifies the number of pixel values that are to be returned in the pixels_return array.

npixels Specifies the number of pixels.

nplanes Specifies the number of plane masks that are to be returned in the plane masks array.

nreds
ngreens
nblues
Specify the number of red, green, and blue planes. The value you pass must be nonnegative.

pixels Specifies an array of pixel values.

pixels_return Returns an array of pixel values.

plane_mask_return
Returns an array of plane masks.

planes Specifies the planes you want to free.

rmask_return
gmask_return
bmask_return Return bit masks for the red, green, and blue planes.

screen_def_return
Returns the closest RGB values provided by the hardware.

screen_in_out Specifies and returns the values actually used in the colormap.

DESCRIPTION

The **XAllocColor** function allocates a read-only colormap entry corresponding to the closest RGB values supported by the hardware. **XAllocColor** returns the pixel value of the color closest to the specified RGB elements supported by the hardware and returns the RGB values actually used. The corresponding colormap cell is read-only. In addition, **XAllocColor** returns nonzero if it succeeded or zero if it failed. Multiple clients that request the same effective RGB values can be assigned the same read-only entry, thus, allowing entries to be shared. When the last client deallocates a shared cell, it is deallocated. **XAllocColor** does not use or affect the flags in the **XColor** structure.

XAllocColor can generate a **BadColor** error.

The **XAllocNamedColor** function looks up the named color with respect to the screen that is associated with the specified colormap. It returns both the exact database definition and the closest color supported by the screen. The allocated color cell is read-only. You should use the ISO Latin-1 encoding; uppercase and lowercase do not matter.

XAllocNamedColor can generate a **BadColor** error.

The **XAllocColorCells** function allocates read/write color cells. The number of colors must be positive and the number of planes nonnegative, or a **BadValue** error results. If ncolors and nplanes are requested, then ncolors pixels and nplane plane masks are returned. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. By ORing together each pixel with zero or more masks, ncolors * $2^{nplanes}$ distinct pixels can be produced. All of these are allocated writable by the request. For **GrayScale** or **PseudoColor**, each mask has exactly one bit set to 1. For **DirectColor**, each has exactly three bits set to 1. If contig is **True** and if all masks are ORed together, a single contiguous set of bits set to 1 will be formed for **GrayScale** or **PseudoColor** and three contiguous sets of bits set to 1 (one within each pixel subfield) for **DirectColor**. The RGB values of the allocated entries are undefined. **XAllocColorCells** returns nonzero if it succeeded or zero if it failed.

XAllocColorCells can generate **BadColor** and **BadValue** errors.

The specified ncolors must be positive; and nreds, ngreens, and nblues must be nonnegative, or a **BadValue** error results. If ncolors colors, nreds reds, ngreens greens, and nblues blues are

requested, ncolors pixels are returned; and the masks have nreds, ngreens, and nblues bits set to 1, respectively. If contig is **True**, each mask will have a contiguous set of bits set to 1. No mask will have any bits set to 1 in common with any other mask or with any of the pixels. For **DirectColor**, each mask will lie within the corresponding pixel subfield. By ORing together subsets of masks with each pixel value, ncolors * $2^{(nreds + ngreens + nblues)}$ distinct pixel values can be produced. All of these are allocated by the request. However, in the colormap, there are only ncolors * 2^{nreds} independent red entries, ncolors * $2^{ngreens}$ independent green entries, and ncolors * 2^{nblues} independent blue entries. This is true even for **PseudoColor**. When the colormap entry of a pixel value is changed (using **XStoreColors**, **XStoreColor**, or **XStoreNamedColor**), the pixel is decomposed according to the masks, and the corresponding independent entries are updated. **XAllocColorPlanes** returns nonzero if it succeeded or zero if it failed.

XAllocColorPlanes can generate **BadColor** and **BadValue** errors.

The **XFreeColors** function frees the cells represented by pixels whose values are in the pixels array. The planes argument should not have any bits set to 1 in common with any of the pixels. The set of all pixels is produced by ORing together subsets of the planes argument with the pixels. The request frees all of these pixels that were allocated by the client (using **XAllocColor**, **XAllocNamedColor**, **XAllocColorCells**, and **XAllocColorPlanes**). Note that freeing an individual pixel obtained from **XAllocColorPlanes** may not actually allow it to be reused until all of its related pixels are also freed. Similarly, a read-only entry is not actually freed until it has been freed by all clients, and if a client allocates the same read-only entry multiple times, it must free the entry that many times before the entry is actually freed.

All specified pixels that are allocated by the client in the colormap are freed, even if one or more pixels produce an error. If a specified pixel is not a valid index into the colormap, a **BadValue** error results. If a specified pixel is not allocated by the client (that is, is unallocated or is only allocated by another client), a **BadAccess** error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XFreeColors can generate **BadAccess**, **BadColor**, and **BadValue** errors.

DIAGNOSTICS

BadAccess	A client attempted to free a color map entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only color map entry.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCreateColormap(3), XQueryColor(3), XStoreColors(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllocIconSize, XSetIconSizes, XGetIconSizes, XIconSize – allocate icon size structure and set or read a window's WM_ICON_SIZES property

SYNTAX

XIconSize *XAllocIconSize()

XSetIconSizes(*display, w, size_list, count*)
Display **display*;
Window *w*;
XIconSize **size_list*;
int *count*;

Status XGetIconSizes(*display, w, size_list_return, count_return*)
Display **display*;
Window *w*;
XIconSize ***size_list_return*;
int **count_return*;

ARGUMENTS

display	Specifies the connection to the X server.
count	Specifies the number of items in the size list.
count_return	Returns the number of items in the size list.
size_list	Specifies a pointer to the size list.
size_list_return	Returns a pointer to the size list.
w	Specifies the window.

DESCRIPTION

The **XAllocIconSize** function allocates and returns a pointer to a **XIconSize** structure. Note that all fields in the **XIconSize** structure are initially set to zero. If insufficient memory is available, **XAllocIconSize** returns NULL. To free the memory allocated to this structure, use **XFree**.

The **XSetIconSizes** function is used only by window managers to set the supported icon sizes.

XSetIconSizes can generate **BadAlloc** and **BadWindow** errors.

The **XGetIconSizes** function returns zero if a window manager has not set icon sizes or nonzero otherwise. **XGetIconSizes** should be called by an application that wants to find out what icon sizes would be most appreciated by the window manager under which the application is running. The application should then use **XSetWMHints** to supply the window manager with an icon pixmap or window in one of the supported sizes. To free the data allocated in size_list_return, use **XFree**.

XGetIconSizes can generate a **BadWindow** error.

PROPERTIES

WM_ICON_SIZES
The window manager may set this property on the root window to specify the icon sizes it supports. The C type of this property is **XIconSize**.

STRUCTURES

The **XIconSize** structure contains:

```
typedef struct {
        int min_width, min_height;
        int max_width, max_height;
        int width_inc, height_inc;
} XIconSize;
```

The width_inc and height_inc members define an arithmetic progression of sizes (minimum to maximum) that represent the supported icon sizes.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllocSizeHints, XSetWMNormalHints, XGetWMNormalHints, XSetWMSizeHints, XGetWMSizeHints, XSizeHints – allocate size hints structure and set or read a window's WM_NORMAL_HINTS property

SYNTAX

XSizeHints *XAllocSizeHints( )

void XSetWMNormalHints(*display, w, hints*)
Display **display*;
Window *w*;
XSizeHints **hints*;

Status XGetWMNormalHints(*display, w, hints_return, supplied_return*)
Display **display*;
Window *w*;
XSizeHints **hints_return*;
long **supplied_return*;

void XSetWMSizeHints(*display, w, hints, property*)
Display **display*;
Window *w*;
XSizeHints **hints*;
Atom *property*;

Status XGetWMSizeHints(*display, w, hints_return, supplied_return, property*)
Display **display*;
Window *w*;
XSizeHints **hints_return*;
long **supplied_return*;
Atom *property*;

ARGUMENTS

display	Specifies the connection to the X server.
hints	Specifies the size hints for the window in its normal state.
hints	Specifies the **XSizeHints** structure to be used.
hints_return	Returns the size hints for the window in its normal state.
property	Specifies the property name.
supplied_return	Returns the hints that were supplied by the user.
w	Specifies the window.

DESCRIPTION

The **XAllocSizeHints** function allocates and returns a pointer to a **XSizeHints** structure. Note that all fields in the **XSizeHints** structure are initially set to zero. If insufficient memory is available, **XAllocSizeHints** returns NULL. To free the memory allocated to this structure, use **XFree**.

The **XSetWMNormalHints** function replaces the size hints for the WM_NORMAL_HINTS property on the specified window. If the property does not already exist, **XSetWMNormalHints** sets the size hints for the WM_NORMAL_HINTS property on the specified window. The property is stored with a type of WM_SIZE_HINTS and a format of 32.

XSetWMNormalHints can generate **BadAlloc** and **BadWindow** errors.

The **XGetWMNormalHints** function returns the size hints stored in the WM_NORMAL_HINTS property on the specified window. If the property is of type WM_SIZE_HINTS, of format 32, and is long enough to contain either an old (pre-ICCCM) or new

size hints structure, **XGetWMNormalHints** sets the various fields of the **XSizeHints** structure, sets the supplied_return argument to the list of fields that were supplied by the user (whether or not they contained defined values) and returns a non-zero status. Otherwise, it returns a zero status.

If **XGetWMNormalHints** returns successfully and a pre-ICCCM size hints property is read, the supplied_return argument will contain the following bits:

(USPosition|USSize|PPosition|PSize|PMinSize|
PMaxSize|PResizeInc|PAspect)

If the property is large enough to contain the base size and window gravity fields as well, the supplied_return argument will also contain the following bits:

PBaseSize|PWinGravity

XGetWMNormalHints can generate a PN BadWindow error.

The **XSetWMSizeHints** function replaces the size hints for the specified property on the named window. If the specified property does not already exist, **XSetWMSizeHints** sets the size hints for the specified property on the named window. The property is stored with a type of WM_SIZE_HINTS and a format of 32. To set a window's normal size hints, you can use the **XSetWMNormalHints** function.

XSetWMSizeHints can generate **BadAlloc**, **BadAtom**, and **BadWindow** errors.

The **XGetWMSizeHints** function returns the size hints stored in the specified property on the named window. If the property is of type WM_SIZE_HINTS, of format 32, and is long enough to contain either an old (pre-ICCCM) or new size hints structure, **XGetWMSizeHints** sets the various fields of the **XSizeHints** structure, sets the supplied_return argument to the list of fields that were supplied by the user (whether or not they contained defined values), and returns a non-zero status. Otherwise, it returns a zero status. To get a window's normal size hints, you can use the **XGetWMNormalHints** function.

If **XGetWMSizeHints** returns successfully and a pre-ICCCM size hints property is read, the supplied_return argument will contain the following bits:

(USPosition|USSize|PPosition|PSize|PMinSize|
PMaxSize|PResizeInc|PAspect)

If the property is large enough to contain the base size and window gravity fields as well, the supplied_return argument will also contain the following bits:

PBaseSize|PWinGravity

XGetWMSizeHints can generate **BadAtom** and **BadWindow** errors.

PROPERTIES

WM_NORMAL_HINTS
: Size hints for a window in its normal state. The C type of this property is **XSizeHints**.

STRUCTURES

The **XSizeHints** structure contains:

```
/* Size hints mask bits */
#define  USPosition   (1L << 0)   /* user specified x, y */
#define  USSize       (1L << 1)   /* user specified width, height */
#define  PPosition    (1L << 2)   /* program specified position */
#define  PSize        (1L << 3)   /* program specified size */
#define  PMinSize     (1L << 4)   /* program specified minimum size */
#define  PMaxSize     (1L << 5)   /* program specified maximum size */
```

```
#define  PResizeInc      (1L << 6)   /* program specified resize increments */
#define  PAspect         (1L << 7)   /* program specified min and max aspect ratios */
#define  PBaseSize       (1L << 8)
#define  PWinGravity     (1L << 9)
#define  PAllHints                   (PPosition|PSize|PMinSize|PMaxSize|
                                      PResizeInc|PAspect)

/* Values */

typedef struct {
        long flags;                  /* marks which fields in this structure are defined */
        int x, y;                    /* Obsolete */
        int width, height;           /* Obsolete */
        int min_width, min_height;
        int max_width, max_height;
        int width_inc, height_inc;
        struct {
             int x;                  /* numerator */
             int y;                  /* denominator */
        } min_aspect, max_aspect;
        int base_width, base_height;
        int win_gravity;
} XSizeHints;
```

The x, y, width, and height members are now obsolete and are left solely for compatibility reasons. The min_width and min_height members specify the minimum window size that still allows the application to be useful. The max_width and max_height members specify the maximum window size. The width_inc and height_inc members define an arithmetic progression of sizes (minimum to maximum) into which the window prefers to be resized. The min_aspect and max_aspect members are expressed as ratios of x and y, and they allow an application to specify the range of aspect ratios it prefers. The base_width and base_height members define the desired size of the window. The win_gravity member defines the region of the window that is to be retained when it is resized.

Note that use of the **PAllHints** macro is highly discouraged.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadAtom A value for an Atom argument does not name a defined Atom.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllocStandardColormap, XSetRGBColormaps, XGetRGBColormaps, XStandardColormap – allocate, set, or read a standard colormap structure

SYNTAX

XStandardColormap *XAllocStandardColormap( )

void XSetRGBColormaps(*display, w, std_colormap, count, property*)
Display **display*;
Window *w*;
XStandardColormap **std_colormap*;
int *count*;
Atom *property*;

Status XGetRGBColormaps(*display, w, std_colormap_return, count_return, property*)
Display **display*;
Window *w*;
XStandardColormap ***std_colormap_return*;
int **count_return*;
Atom *property*;

ARGUMENTS

display	Specifies the connection to the X server.
count	Specifies the number of colormaps.
count_return	Returns the number of colormaps.
property	Specifies the property name.
std_colormap	Specifies the **XStandardColormap** structure to be used.
std_colormap_return	Returns the **XStandardColormap** structure.

DESCRIPTION

The **XAllocStandardColormap** function allocates and returns a pointer to a **XStandardColormap** structure. Note that all fields in the **XStandardColormap** structure are initially set to zero. If insufficient memory is available, **XAllocStandardColormap** returns NULL. To free the memory allocated to this structure, use **XFree**.

The **XSetRGBColormaps** function replaces the RGB colormap definition in the specified property on the named window. If the property does not already exist, **XSetRGBColormaps** sets the RGB colormap definition in the specified property on the named window. The property is stored with a type of RGB_COLOR_MAP and a format of 32. Note that it is the caller's responsibility to honor the ICCCM restriction that only RGB_DEFAULT_MAP contain more than one definition.

XSetRGBColormaps can generate **BadAlloc**, **BadAtom**, and **BadWindow** errors.

The **XGetRGBColormaps** function returns the RGB colormap definitions stored in the specified property on the named window. If the property exists, is of type RGB_COLOR_MAP, is of format 32, and is long enough to contain a colormap definition (if the visualid is not present, **XGetRGBColormaps** assumes the default visual for the screen on which the window is located; if the killid is not present, **None**, which indicates that the resources cannot be released, is assumed), **XGetRGBColormaps** allocates and fills in space for the returned colormaps, and returns a non-zero status. Otherwise, none of the fields are set, and **XGetRGBColormaps** returns a zero status. Note that it is the caller's responsibility to honor the ICCCM restriction that only RGB_DEFAULT_MAP contain more than one definition.

XGetRGBColormaps can generate **BadAtom** and **BadWindow** errors.

STRUCTURES

The **XStandardColormap** structure contains:

```
/* Hints */
#define     ReleaseByFreeingColormap      ( (XID) 1L)
/* Values */
typedef struct {
        Colormap colormap;
        unsigned long red_max;
        unsigned long red_mult;
        unsigned long green_max;
        unsigned long green_mult;
        unsigned long blue_max;
        unsigned long blue_mult;
        unsigned long base_pixel;
        VisualID visualid;
        XID killid;
} XStandardColormap;
```

The colormap member is the colormap created by the **XCreateColormap** function. The red_max, green_max, and blue_max members give the maximum red, green, and blue values, respectively. Each color coefficient ranges from zero to its max, inclusive. For example, a common colormap allocation is 3/3/2 (3 planes for red, 3 planes for green, and 2 planes for blue). This colormap would have red_max = 7, green_max = 7, and blue_max = 3. An alternate allocation that uses only 216 colors is red_max = 5, green_max = 5, and blue_max = 5.

The red_mult, green_mult, and blue_mult members give the scale factors used to compose a full pixel value. (See the discussion of the base_pixel members for further information.) For a 3/3/2 allocation, red_mult might be 32, green_mult might be 4, and blue_mult might be 1. For a 6-colors-each allocation, red_mult might be 36, green_mult might be 6, and blue_mult might be 1.

The base_pixel member gives the base pixel value used to compose a full pixel value. Usually, the base_pixel is obtained from a call to the **XAllocColorPlanes** function. Given integer red, green, and blue coefficients in their appropriate ranges, one then can compute a corresponding pixel value by using the following expression:

r * red_mult + g * green_mult + b * blue_mult + base_pixel

For **GrayScale** colormaps, only the colormap, red_max, red_mult, and base_pixel members are defined. The other members are ignored.

The visualid member gives the ID number of the visual from which the colormap was created. The killid member gives a resource ID that indicates whether the cells held by this standard colormap are to be released by freeing the colormap ID or by calling the **XKillClient** function on the indicated resource. (Note that this method is necessary for allocating out of an existing colormap).

To compute a **GrayScale** pixel value, use the following expression:

gray * red_mult + base_pixel

The properties containing the **XStandardColormap** information have the type RGB_COLOR_MAP.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocColor(3), XCreateColormap(3), XFree(3), XSetCloseDownMode(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllocWMHints, XSetWMHints, XGetWMHints, XWMHints - allocate window manager hints structure and set or read a window's WM_HINTS property

SYNTAX

XWMHints *XAllocWMHints( )

XSetWMHints(*display, w, wmhints*)
Display **display*;
Window *w*;
XWMHints **wmhints*;

XWMHints *XGetWMHints(*display, w*)
Display **display*;
Window *w*;

ARGUMENTS

display	Specifies the connection to the X server.
w	Specifies the window.
wmhints	Specifies the **XWMHints** structure to be used.

DESCRIPTION

The **XAllocWMHints** function allocates and returns a pointer to a **XWMHints** structure. Note that all fields in the **XWMHints** structure are initially set to zero. If insufficient memory is available, **XAllocWMHints** returns NULL. To free the memory allocated to this structure, use **XFree**.

The **XSetWMHints** function sets the window manager hints that include icon information and location, the initial state of the window, and whether the application relies on the window manager to get keyboard input.

XSetWMHints can generate **BadAlloc** and **BadWindow** errors.

The **XGetWMHints** function reads the window manager hints and returns NULL if no WM_HINTS property was set on the window or returns a pointer to a **XWMHints** structure if it succeeds. When finished with the data, free the space used for it by calling **XFree**.

XGetWMHints can generate a **BadWindow** error.

PROPERTIES

WM_HINTS	Additional hints set by client for use by the window manager. The C type of this property is **XWMHints**.

STRUCTURES

The **XWMHints** structure contains:

```
/* Window manager hints mask bits */
#define    InputHint                (1L << 0)
#define    StateHint                (1L << 1)
#define    IconPixmapHint           (1L << 2)
#define    IconWindowHint           (1L << 3)
#define    IconPositionHint         (1L << 4)
#define    IconMaskHint             (1L << 5)
#define    WindowGroupHint          (1L << 6)
#define    AllHints                 (InputHint|StateHint|IconPixmapHint|
                                     IconWindowHint|IconPositionHint|
                                     IconMaskHint|WindowGroupHint)

/* Values */
```

```
typedef struct {
        long flags;                 /* marks which fields in this structure are defined */
        Bool input;                 /* does this application rely on the window manager to
                                       get keyboard input? */
        int initial_state;          /* see below */
        Pixmap icon_pixmap;         /* pixmap to be used as icon */
        Window icon_window;         /* window to be used as icon */
        int icon_x, icon_y;         /* initial position of icon */
        Pixmap icon_mask;           /* pixmap to be used as mask for icon_pixmap */
        XID window_group;           /* id of related window group */
        /* this structure may be extended in the future */
} XWMHints;
```

The input member is used to communicate to the window manager the input focus model used by the application. Applications that expect input but never explicitly set focus to any of their subwindows (that is, use the push model of focus management), such as X10-style applications that use real-estate driven focus, should set this member to **True**. Similarly, applications that set input focus to their subwindows only when it is given to their top-level window by a window manager should also set this member to **True**. Applications that manage their own input focus by explicitly setting focus to one of their subwindows whenever they want keyboard input (that is, use the pull model of focus management) should set this member to **False**. Applications that never expect any keyboard input also should set this member to **False**.

Pull model window managers should make it possible for push model applications to get input by setting input focus to the top-level windows of applications whose input member is **True**. Push model window managers should make sure that pull model applications do not break them by resetting input focus to **PointerRoot** when it is appropriate (for example, whenever an application whose input member is **False** sets input focus to one of its subwindows).

The definitions for the initial_state flag are:

```
#define  WithdrawnState   0
#define  NormalState      1   /* most applications start this way */
#define  IconicState      3   /* application wants to start as an icon */
```

The icon_mask specifies which pixels of the icon_pixmap should be used as the icon. This allows for nonrectangular icons. Both icon_pixmap and icon_mask must be bitmaps. The icon_window lets an application provide a window for use as an icon for window managers that support such use. The window_group lets you specify that this window belongs to a group of other windows. For example, if a single application manipulates multiple top-level windows, this allows you to provide enough information that a window manager can iconify all of the windows rather than just the one window.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAllowEvents – release queued events

SYNTAX

XAllowEvents(*display, event_mode, time*)
Display **display*;
int *event_mode*;
Time *time*;

ARGUMENTS

display	Specifies the connection to the X server.
event_mode	Specifies the event mode. You can pass **AsyncPointer**, **SyncPointer**, **AsyncKeyboard**, **SyncKeyboard**, **ReplayPointer**, **ReplayKeyboard**, **AsyncBoth**, or **SyncBoth**.
time	Specifies the time. You can pass either a timestamp or **CurrentTime**.

DESCRIPTION

The **XAllowEvents** function releases some queued events if the client has caused a device to freeze. It has no effect if the specified time is earlier than the last-grab time of the most recent active grab for the client or if the specified time is later than the current X server time.

XAllowEvents can generate a **BadValue** error.

DIAGNOSTICS

BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XAnyEvent, XEvent – generic X event structures

STRUCTURES

All the event structures declared in <**X11/Xlib.h**> have the following common members:

```
typedef struct {
        int type;
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
} XAnyEvent;
```

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The **XEvent** structure is a union of the individual structures declared for each event type:

```
typedef union _XEvent {
        int type;                       /* must not be changed */
        XAnyEvent xany;
        XKeyEvent xkey;
        XButtonEvent xbutton;
        XMotionEvent xmotion;
        XCrossingEvent xcrossing;
        XFocusChangeEvent xfocus;
        XExposeEvent xexpose;
        XGraphicsExposeEvent xgraphicsexpose;
        XNoExposeEvent xnoexpose;
        XVisibilityEvent xvisibility;
        XCreateWindowEvent xcreatewindow;
        XDestroyWindowEvent xdestroywindow;
        XUnmapEvent xunmap;
        XMapEvent xmap;
        XMapRequestEvent xmaprequest;
        XReparentEvent xreparent;
        XConfigureEvent xconfigure;
        XGravityEvent xgravity;
        XResizeRequestEvent xresizerequest;
        XConfigureRequestEvent xconfigurerequest;
        XCirculateEvent xcirculate;
        XCirculateRequestEvent xcirculaterequest;
        XPropertyEvent xproperty;
        XSelectionClearEvent xselectionclear;
        XSelectionRequestEvent xselectionrequest;
        XSelectionEvent xselection;
        XColormapEvent xcolormap;
        XClientMessageEvent xclient;
        XMappingEvent xmapping;
        XErrorEvent xerror;
        XKeymapEvent xkeymap;
```

```
        long pad[24];
} XEvent;
```

An **XEvent** structure's first entry always is the type member, which is set to the event type. The second member always is the serial number of the protocol request that generated the event. The third member always is send_event, which is a **Bool** that indicates if the event was sent by a different client. The fourth member always is a display, which is the display that the event was read from. Except for keymap events, the fifth member always is a window, which has been carefully selected to be useful to toolkit dispatchers. To avoid breaking toolkits, the order of these first five entries is not to change. Most events also contain a time member, which is the time at which an event occurred. In addition, a pointer to the generic event must be cast before it is used to access any other information in the structure.

SEE ALSO

XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XButtonEvent, XKeyEvent, XMotionEvent – KeyPress, KeyRelease, ButtonPress, ButtonRelease, and MotionNotify event structures

STRUCTURES

The structures for **KeyPress**, **KeyRelease**, **ButtonPress**, **ButtonRelease**, and **MotionNotify** events contain:

```
typedef struct {
        int type;                       /* ButtonPress or ButtonRelease */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;                  /* "event" window it is reported relative to */
        Window root;                    /* root window that the event occurred on */
        Window subwindow;               /* child window */
        Time time;                      /* milliseconds */
        int x, y;                       /* pointer x, y coordinates in event window */
        int x_root, y_root;             /* coordinates relative to root */
        unsigned int state;             /* key or button mask */
        unsigned int button;            /* detail */
        Bool same_screen;               /* same screen flag */
} XButtonEvent;
typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;

typedef struct {
        int type;                       /* KeyPress or KeyRelease */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;                  /* "event" window it is reported relative to */
        Window root;                    /* root window that the event occurred on */
        Window subwindow;               /* child window */
        Time time;                      /* milliseconds */
        int x, y;                       /* pointer x, y coordinates in event window */
        int x_root, y_root;             /* coordinates relative to root */
        unsigned int state;             /* key or button mask */
        unsigned int keycode;           /* detail */
        Bool same_screen;               /* same screen flag */
} XKeyEvent;
typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;

typedef struct {
        int type;                       /* MotionNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;                  /* "event" window reported relative to */
        Window root;                    /* root window that the event occurred on */
        Window subwindow;               /* child window */
        Time time;                      /* milliseconds */
        int x, y;                       /* pointer x, y coordinates in event window */
        int x_root, y_root;             /* coordinates relative to root */
        unsigned int state;             /* key or button mask */
```

```
    char is_hint;                    /* detail */
    Bool same_screen;                /* same screen flag */
} XMotionEvent;
typedef XMotionEvent XPointerMovedEvent;
```

When you receive these events, their structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

These structures have the following common members: window, root, subwindow, time, x, y, x_root, y_root, state, and same_screen. The window member is set to the window on which the event was generated and is referred to as the event window. As long as the conditions previously discussed are met, this is the window used by the X server to report the event. The root member is set to the source window's root window. The x_root and y_root members are set to the pointer's coordinates relative to the root window's origin at the time of the event.

The same_screen member is set to indicate whether the event window is on the same screen as the root window and can be either **True** or **False**. If **True**, the event and root windows are on the same screen. If **False**, the event and root windows are not on the same screen.

If the source window is an inferior of the event window, the subwindow member of the structure is set to the child of the event window that is the source member or an ancestor of it. Otherwise, the X server sets the subwindow member to **None**. The time member is set to the time when the event was generated and is expressed in milliseconds.

If the event window is on the same screen as the root window, the x and y members are set to the coordinates relative to the event window's origin. Otherwise, these members are set to zero.

The state member is set to indicate the logical state of the pointer buttons and modifier keys just prior to the event, which is the bitwise inclusive OR of one or more of the button or modifier key masks: **Button1Mask**, **Button2Mask**, **Button3Mask**, **Button4Mask**, **Button5Mask**, **ShiftMask**, **LockMask**, **ControlMask**, **Mod1Mask**, **Mod2Mask**, **Mod3Mask**, **Mod4Mask**, and **Mod5Mask**.

Each of these structures also has a member that indicates the detail. For the **XKeyPressedEvent** and **XKeyReleasedEvent** structures, this member is called keycode. It is set to a number that represents a physical key on the keyboard. The keycode is an arbitrary representation for any key on the keyboard (see chapter 7).

For the **XButtonPressedEvent** and **XButtonReleasedEvent** structures, this member is called button. It represents the pointer button that changed state and can be the **Button1**, **Button2**, **Button3**, **Button4**, or **Button5** value. For the **XPointerMovedEvent** structure, this member is called is_hint. It can be set to **NotifyNormal** or **NotifyHint**.

SEE ALSO

XAnyEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XChangeKeyboardControl, XGetKeyboardControl, XAutoRepeatOn, XAutoRepeatOff, XBell, XQueryKeymap, XKeyboardControl – manipulate keyboard settings and keyboard control structure

SYNTAX

XChangeKeyboardControl(*display, value_mask, values*)
Display **display*;
unsigned long *value_mask*;
XKeyboardControl **values*;

XGetKeyboardControl(*display, values_return*)
Display **display*;
XKeyboardState **values_return*;

XAutoRepeatOn(*display*)
Display **display*;

XAutoRepeatOff(*display*)
Display **display*;

XBell(*display, percent*)
Display **display*;
int *percent*;

XQueryKeymap(*display, keys_return*)
Display **display*;
char *keys_return*[32];

ARGUMENTS

display	Specifies the connection to the X server.
keys_return	Returns an array of bytes that identifies which keys are pressed down. Each bit represents one key of the keyboard.
percent	Specifies the volume for the bell, which can range from –100 to 100 inclusive.
value_mask	Specifies one value for each bit set to 1 in the mask.
values	Specifies which controls to change. This mask is the bitwise inclusive OR of the valid control mask bits.
values_return	Returns the current keyboard controls in the specified **XKeyboardState** structure.

DESCRIPTION

The **XChangeKeyboardControl** function controls the keyboard characteristics defined by the **XKeyboardControl** structure. The value_mask argument specifies which values are to be changed.

XChangeKeyboardControl can generate **BadMatch** and **BadValue** errors.

The **XGetKeyboardControl** function returns the current control values for the keyboard to the **XKeyboardState** structure.

The **XAutoRepeatOn** function turns on auto-repeat for the keyboard on the specified display.

The **XAutoRepeatOff** function turns off auto-repeat for the keyboard on the specified display.

The **XBell** function rings the bell on the keyboard on the specified display, if possible. The specified volume is relative to the base volume for the keyboard. If the value for the percent argument is not in the range –100 to 100 inclusive, a **BadValue** error results. The volume at which the bell rings when the percent argument is nonnegative is:

base – [(base * percent) / 100] + percent

The volume at which the bell rings when the percent argument is negative is:

base + [(base * percent) / 100]

To change the base volume of the bell, use **XChangeKeyboardControl**.

XBell can generate a **BadValue** error.

The **XQueryKeymap** function returns a bit vector for the logical state of the keyboard, where each bit set to 1 indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N.

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

STRUCTURES

The **XKeyboardControl** structure contains:

```
/* Mask bits for ChangeKeyboardControl */
#define    KBKeyClickPercent            (1L<<0)
#define    KBBellPercent                (1L<<1)
#define    KBBellPitch                  (1L<<2)
#define    KBBellDuration               (1L<<3)
#define    KBLed                        (1L<<4)
#define    KBLedMode                    (1L<<5)
#define    KBKey                        (1L<<6)
#define    KBAutoRepeatMode             (1L<<7)
/* Values */

typedef struct {
        int key_click_percent;
        int bell_percent;
        int bell_pitch;
        int bell_duration;
        int led;
        int led_mode;              /* LedModeOn, LedModeOff */
        int key;
        int auto_repeat_mode;      /* AutoRepeatModeOff, AutoRepeatModeOn,
                                      AutoRepeatModeDefault */
} XKeyboardControl;
```

The key_click_percent member sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate a **BadValue** error.

The bell_percent sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. A setting of -1 restores the default. Other negative values generate a **BadValue** error. The bell_pitch member sets the pitch (specified in Hz) of the bell, if possible. A setting of -1 restores the default. Other negative values generate a **BadValue** error. The bell_duration member sets the duration of the bell specified in milliseconds, if possible. A setting of -1 restores the default. Other negative values generate a **BadValue** error.

If both the led_mode and led members are specified, the state of that LED is changed, if possible. The led_mode member can be set to **LedModeOn** or **LedModeOff**. If only led_mode is specified, the state of all LEDs are changed, if possible. At most 32 LEDs numbered from one are supported. No standard interpretation of LEDs is defined. If led is specified without led_mode, a **BadMatch** error results.

If both the auto_repeat_mode and key members are specified, the auto_repeat_mode of that key is changed (according to **AutoRepeatModeOn**, **AutoRepeatModeOff**, or **AutoRepeatModeDefault**), if possible. If only auto_repeat_mode is specified, the global auto_repeat_mode for the entire keyboard is changed, if possible, and does not affect the per key settings. If a key is specified without an auto_repeat_mode, a **BadMatch** error results. Each key has an individual mode of whether or not it should auto-repeat and a default setting for the mode. In addition, there is a global mode of whether auto-repeat should be enabled or not and a default setting for that mode. When global mode is **AutoRepeatModeOn**, keys should obey their individual auto-repeat modes. When global mode is **AutoRepeatModeOff**, no keys should auto-repeat. An auto-repeating key generates alternating **KeyPress** and **KeyRelease** events. When a key is used as a modifier, it is desirable for the key not to auto-repeat, regardless of its auto-repeat setting.

The **XKeyboardState** structure contains:

```
typedef struct {
        int key_click_percent;
        int bell_percent;
        unsigned int bell_pitch, bell_duration;
        unsigned long led_mask;
        int global_auto_repeat;
        char auto_repeats[32];
} XKeyboardState;
```

For the LEDs, the least-significant bit of led_mask corresponds to LED one, and each bit set to 1 in led_mask indicates an LED that is lit. The global_auto_repeat member can be set to **AutoRepeatModeOn** or **AutoRepeatModeOff**. The auto_repeats member is a bit vector. Each bit set to 1 indicates that auto-repeat is enabled for the corresponding key. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N.

DIAGNOSTICS

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XChangeKeyboardMapping(3), XSetPointerMapping(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XChangeKeyboardMapping, XGetKeyboardMapping, XDisplayKeycodes, XSetModifierMapping, XGetModifierMapping, XNewModifiermap, XInsertModifiermapEntry, XDeleteModifiermapEntry, XFreeModifierMap XModifierKeymap – manipulate keyboard encoding and keyboard encoding structure

SYNTAX

XChangeKeyboardMapping(*display, first_keycode, keysyms_per_keycode, keysyms, num_codes*)
Display **display*;
int *first_keycode*;
int *keysyms_per_keycode*;
KeySym **keysyms*;
int *num_codes*;

KeySym *XGetKeyboardMapping(*display, first_keycode, keycode_count, keysyms_per_keycode_return*)
Display **display*;
KeyCode *first_keycode*;
int *keycode_count*;
int **keysyms_per_keycode_return*;

XDisplayKeycodes(*display, min_keycodes_return, max_keycodes_return*)
Display **display*;
int **min_keycodes_return*, **max_keycodes_return*;

int XSetModifierMapping(*display, modmap*)
Display **display*;
XModifierKeymap **modmap*;

XModifierKeymap *XGetModifierMapping(*display*)
Display **display*;

XModifierKeymap *XNewModifiermap(*max_keys_per_mod*)
int *max_keys_per_mod*;

XModifierKeymap *XInsertModifiermapEntry(*modmap, keycode_entry, modifier*)
XModifierKeymap **modmap*;
KeyCode *keycode_entry*;
int *modifier*;

XModifierKeymap *XDeleteModifiermapEntry(*modmap, keycode_entry, modifier*)
XModifierKeymap **modmap*;
KeyCode *keycode_entry*;
int *modifier*;

XFreeModifiermap(*modmap*)
XModifierKeymap **modmap*;

ARGUMENTS

display	Specifies the connection to the X server.
first_keycode	Specifies the first KeyCode that is to be changed or returned.
keycode_count	Specifies the number of KeyCodes that are to be returned.
keycode_entry	Specifies the KeyCode.
keysyms	Specifies a pointer to an array of KeySyms.
keysyms_per_keycode	Specifies the number of KeySyms per KeyCode.

keysyms_per_keycode_return
Returns the number of KeySyms per KeyCode.

max_keys_per_mod
Specifies the number of KeyCode entries preallocated to the modifiers in the map.

max_keycodes_return
Returns the maximum number of KeyCodes.

min_keycodes_return
Returns the minimum number of KeyCodes.

modifier Specifies the modifier.

modmap Specifies a pointer to the **XModifierKeymap** structure.

num_codes Specifies the number of KeyCodes that are to be changed.

DESCRIPTION

The **XChangeKeyboardMapping** function defines the symbols for the specified number of KeyCodes starting with first_keycode. The symbols for KeyCodes outside this range remain unchanged. The number of elements in keysyms must be:

num_codes * keysyms_per_keycode

The specified first_keycode must be greater than or equal to min_keycode returned by **XDisplayKeycodes**, or a **BadValue** error results. In addition, the following expression must be less than or equal to max_keycode as returned by **XDisplayKeycodes**, or a **BadValue** error results:

first_keycode + num_codes - 1

KeySym number N, counting from zero, for KeyCode K has the following index in keysyms, counting from zero:

(K - first_keycode) * keysyms_per_keycode + N

The specified keysyms_per_keycode can be chosen arbitrarily by the client to be large enough to hold all desired symbols. A special KeySym value of **NoSymbol** should be used to fill in unused elements for individual KeyCodes. It is legal for **NoSymbol** to appear in nontrailing positions of the effective list for a KeyCode. **XChangeKeyboardMapping** generates a **MappingNotify** event.

There is no requirement that the X server interpret this mapping. It is merely stored for reading and writing by clients.

XChangeKeyboardMapping can generate **BadAlloc** and **BadValue** errors.

The **XGetKeyboardMapping** function returns the symbols for the specified number of KeyCodes starting with first_keycode. The value specified in first_keycode must be greater than or equal to min_keycode as returned by **XDisplayKeycodes**, or a **BadValue** error results. In addition, the following expression must be less than or equal to max_keycode as returned by **XDisplayKeycodes**:

first_keycode + keycode_count - 1

If this is not the case, a **BadValue** error results. The number of elements in the KeySyms list is:

keycode_count * keysyms_per_keycode_return

KeySym number N, counting from zero, for KeyCode K has the following index in the list, counting from zero:
(K - first_code) * keysyms_per_code_return + N

The X server arbitrarily chooses the keysyms_per_keycode_return value to be large enough to report all requested symbols. A special KeySym value of **NoSymbol** is used to fill in unused elements for individual KeyCodes. To free the storage returned by **XGetKeyboardMapping**, use **XFree**.

XGetKeyboardMapping can generate a **BadValue** error.

The **XDisplayKeycodes** function returns the min-keycodes and max-keycodes supported by the specified display. The minimum number of KeyCodes returned is never less than 8, and the maximum number of KeyCodes returned is never greater than 255. Not all KeyCodes in this range are required to have corresponding keys.

The **XSetModifierMapping** function specifies the KeyCodes of the keys (if any) that are to be used as modifiers. If it succeeds, the X server generates a **MappingNotify** event, and **XSetModifierMapping** returns **MappingSuccess**. X permits at most eight modifier keys. If more than eight are specified in the **XModifierKeymap** structure, a **BadLength** error results.

The modifiermap member of the **XModifierKeymap** structure contains eight sets of max_keypermod KeyCodes, one for each modifier in the order **Shift**, **Lock**, **Control**, **Mod1**, **Mod2**, **Mod3**, **Mod4**, and **Mod5**. Only nonzero KeyCodes have meaning in each set, and zero KeyCodes are ignored. In addition, all of the nonzero KeyCodes must be in the range specified by min_keycode and max_keycode in the **Display** structure, or a **BadValue** error results. No KeyCode may appear twice in the entire map, or a **BadValue** error results.

An X server can impose restrictions on how modifiers can be changed, for example, if certain keys do not generate up transitions in hardware, if auto-repeat cannot be disabled on certain keys, or if multiple modifier keys are not supported. If some such restriction is violated, the status reply is **MappingFailed**, and none of the modifiers are changed. If the new KeyCodes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are in the logically down state, **XSetModifierMapping** returns **MappingBusy**, and none of the modifiers is changed.

XSetModifierMapping can generate **BadAlloc** and **BadValue** errors.

The **XGetModifierMapping** function returns a pointer to a newly created **XModifierKeymap** structure that contains the keys being used as modifiers. The structure should be freed after use by calling **XFreeModifiermap**. If only zero values appear in the set for any modifier, that modifier is disabled.

The **XNewModifiermap** function returns a pointer to **XModifierKeymap** structure for later use.

The **XInsertModifiermapEntry** function adds the specified KeyCode to the set that controls the specified modifier and returns the resulting **XModifierKeymap** structure (expanded as needed).

The **XDeleteModifiermapEntry** function deletes the specified KeyCode from the set that controls the specified modifier and returns a pointer to the resulting **XModifierKeymap** structure.

The **XFreeModifiermap** function frees the specified **XModifierKeymap** structure.

STRUCTURES

The **XModifierKeymap** structure contains:

```
typedef struct {
        int max_keypermod;          /* This server's max number of keys per modifier */
        KeyCode *modifiermap;       /* An 8 by max_keypermod array of the modifiers */
} XModifierKeymap;
```

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XFree(3), XSetPointerMapping(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XChangePointerControl, XGetPointerControl – control pointer

SYNTAX

XChangePointerControl(*display, do_accel, do_threshold, accel_numerator, accel_denominator, threshold*)

Display **display*;
Bool *do_accel, do_threshold*;
int *accel_numerator, accel_denominator*;
int *threshold*;

XGetPointerControl(*display, accel_numerator_return, accel_denominator_return, threshold_return*)

Display **display*;
int **accel_numerator_return*, **accel_denominator_return*;
int **threshold_return*;

ARGUMENTS

accel_denominator
Specifies the denominator for the acceleration multiplier.

accel_denominator_return
Returns the denominator for the acceleration multiplier.

accel_numerator
Specifies the numerator for the acceleration multiplier.

accel_numerator_return
Returns the numerator for the acceleration multiplier.

display Specifies the connection to the X server.

do_accel Specifies a Boolean value that controls whether the values for the accel_numerator or accel_denominator are used.

do_threshold Specifies a Boolean value that controls whether the value for the threshold is used.

threshold Specifies the acceleration threshold.

threshold_return
Returns the acceleration threshold.

DESCRIPTION

The **XChangePointerControl** function defines how the pointing device moves. The acceleration, expressed as a fraction, is a multiplier for movement. For example, specifying 3/1 means the pointer moves three times as fast as normal. The fraction may be rounded arbitrarily by the X server. Acceleration only takes effect if the pointer moves more than threshold pixels at once and only applies to the amount beyond the value in the threshold argument. Setting a value to –1 restores the default. The values of the do_accel and do_threshold arguments must be **True** for the pointer values to be set, or the parameters are unchanged. Negative values (other than –1) generate a **BadValue** error, as does a zero value for the accel_denominator argument.

XChangePointerControl can generate a **BadValue** error.

The **XGetPointerControl** function returns the pointer's current acceleration multiplier and acceleration threshold.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can

generate this error.

SEE ALSO

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XChangeSaveSet, XAddToSaveSet, XRemoveFromSaveSet – change a client's save set

SYNTAX

XChangeSaveSet(*display*, *w*, *change_mode*)
Display **display*;
Window *w*;
int *change_mode*;

XAddToSaveSet(*display*, *w*)
Display **display*;
Window *w*;

XRemoveFromSaveSet(*display*, *w*)
Display **display*;
Window *w*;

ARGUMENTS

change_mode	Specifies the mode. You can pass **SetModeInsert** or **SetModeDelete**.
display	Specifies the connection to the X server.
w	Specifies the window that you want to add or delete from the client's save-set.

DESCRIPTION

Depending on the specified mode, **XChangeSaveSet** either inserts or deletes the specified window from the client's save-set. The specified window must have been created by some other client, or a **BadMatch** error results.

XChangeSaveSet can generate **BadMatch**, **BadValue**, and **BadWindow** errors.

The **XAddToSaveSet** function adds the specified window to the client's save-set. The specified window must have been created by some other client, or a **BadMatch** error results.

XAddToSaveSet can generate **BadMatch** and **BadWindow** errors.

The **XRemoveFromSaveSet** function removes the specified window from the client's save-set. The specified window must have been created by some other client, or a **BadMatch** error results.

XRemoveFromSaveSet can generate **BadMatch** and **BadWindow** errors.

DIAGNOSTICS

BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XReparentWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XChangeWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap – change window attributes

SYNTAX

XChangeWindowAttributes(*display, w, valuemask, attributes*)
Display **display*;
Window *w*;
unsigned long *valuemask*;
XSetWindowAttributes **attributes*;

XSetWindowBackground(*display, w, background_pixel*)
Display **display*;
Window *w*;
unsigned long *background_pixel*;

XSetWindowBackgroundPixmap(*display, w, background_pixmap*)
Display **display*;
Window *w*;
Pixmap *background_pixmap*;

XSetWindowBorder(*display, w, border_pixel*)
Display **display*;
Window *w*;
unsigned long *border_pixel*;

XSetWindowBorderPixmap(*display, w, border_pixmap*)
Display **display*;
Window *w*;
Pixmap *border_pixmap*;

ARGUMENTS

attributes Specifies the structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure.

background_pixel Specifies the pixel that is to be used for the background.

background_pixmap Specifies the background pixmap, **ParentRelative**, or **None**.

border_pixel Specifies the entry in the colormap.

border_pixmap Specifies the border pixmap or **CopyFromParent**.

display Specifies the connection to the X server.

valuemask Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced.

w Specifies the window.

DESCRIPTION

Depending on the valuemask, the **XChangeWindowAttributes** function uses the window attributes in the **XSetWindowAttributes** structure to change the specified window attributes. Changing the background does not cause the window contents to be changed. To repaint the window and its background, use **XClearWindow**. Setting the border or changing the background such that the border tile origin changes causes the border to be repainted. Changing the background of a root window to **None** or **ParentRelative** restores the default background pixmap. Changing the border of a root window to **CopyFromParent** restores the default border

pixmap. Changing the win-gravity does not affect the current position of the window. Changing the backing-store of an obscured window to **WhenMapped** or **Always**, or changing the backing-planes, backing-pixel, or save-under of a mapped window may have no immediate effect. Changing the colormap of a window (that is, defining a new map, not changing the contents of the existing map) generates a **ColormapNotify** event. Changing the colormap of a visible window may have no immediate effect on the screen because the map may not be installed (see **XInstallColormap**). Changing the cursor of a root window to **None** restores the default cursor. Whenever possible, you are encouraged to share colormaps.

Multiple clients can select input on the same window. Their event masks are maintained separately. When an event is generated, it is reported to all interested clients. However, only one client at a time can select for **SubstructureRedirectMask**, **ResizeRedirectMask**, and **ButtonPressMask**. If a client attempts to select any of these event masks and some other client has already selected one, a **BadAccess** error results. There is only one do-not-propagate-mask for a window, not one per client.

XChangeWindowAttributes can generate **BadAccess**, **BadColor**, **BadCursor**, **BadMatch**, **BadPixmap**, **BadValue**, and **BadWindow** errors.

The **XSetWindowBackground** function sets the background of the window to the specified pixel value. Changing the background does not cause the window contents to be changed. **XSetWindowBackground** uses a pixmap of undefined size filled with the pixel value you passed. If you try to change the background of an **InputOnly** window, a **BadMatch** error results.

XSetWindowBackground can generate **BadMatch** and **BadWindow** errors.

The **XSetWindowBackgroundPixmap** function sets the background pixmap of the window to the specified pixmap. The background pixmap can immediately be freed if no further explicit references to it are to be made. If **ParentRelative** is specified, the background pixmap of the window's parent is used, or on the root window, the default background is restored. If you try to change the background of an **InputOnly** window, a **BadMatch** error results. If the background is set to **None**, the window has no defined background.

XSetWindowBackgroundPixmap can generate **BadMatch**, **BadPixmap**, and **BadWindow** errors.

The **XSetWindowBorder** function sets the border of the window to the pixel value you specify. If you attempt to perform this on an **InputOnly** window, a **BadMatch** error results.

XSetWindowBorder can generate **BadMatch** and **BadWindow** errors.

The **XSetWindowBorderPixmap** function sets the border pixmap of the window to the pixmap you specify. The border pixmap can be freed immediately if no further explicit references to it are to be made. If you specify **CopyFromParent**, a copy of the parent window's border pixmap is used. If you attempt to perform this on an **InputOnly** window, a **BadMatch** error results.

XSetWindowBorderPixmap can generate **BadMatch**, **BadPixmap**, and **BadWindow** errors.

DIAGNOSTICS

BadAccess	A client attempted to free a color map entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only color map entry.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadMatch An **InputOnly** window locks this attribute.

BadPixmap A value for a Pixmap argument does not name a defined Pixmap.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XConfigureWindow(3), XCreateWindow(3), XDestroyWindow(3), XInstallColormap(3), XMapWindow(3), XRaiseWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCirculateEvent – CirculateNotify event structure

STRUCTURES

The structure for **CirculateNotify** events contains:

```
typedef struct {
        int type;                       /* CirculateNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
        int place;                      /* PlaceOnTop, PlaceOnBottom */
} XCirculateEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the restacked window or to its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. The window member is set to the window that was restacked. The place member is set to the window's position after the restack occurs and is either **PlaceOnTop** or **PlaceOnBottom**. If it is **PlaceOnTop**, the window is now on top of all siblings. If it is **PlaceOnBottom**, the window is now below all siblings.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCirculateRequestEvent – CirculateRequest event structure

STRUCTURES

The structure for **CirculateRequest** events contains:

```
typedef struct {
        int type;                       /* CirculateRequest */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window parent;
        Window window;
        int place;                      /* PlaceOnTop, PlaceOnBottom */
} XCirculateRequestEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The parent member is set to the parent window. The window member is set to the subwindow to be restacked. The place member is set to what the new position in the stacking order should be and is either **PlaceOnTop** or **PlaceOnBottom**. If it is **PlaceOnTop**, the subwindow should be on top of all siblings. If it is **PlaceOnBottom**, the subwindow should be below all siblings.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XClearArea, XClearWindow - clear area or window

SYNTAX

XClearArea(*display, w, x, y, width, height, exposures*)
Display **display*;
Window *w*;
int *x, y*;
unsigned int *width, height*;
Bool *exposures*;

XClearWindow(*display, w*)
Display **display*;
Window *w*;

ARGUMENTS

display	Specifies the connection to the X server.
exposures	Specifies a Boolean value that indicates if **Expose** events are to be generated.
w	Specifies the window.
width *height*	Specify the width and height, which are the dimensions of the rectangle.
x *y*	Specify the x and y coordinates, which are relative to the origin of the window and specify the upper-left corner of the rectangle.

DESCRIPTION

The **XClearArea** function paints a rectangular area in the specified window according to the specified dimensions with the window's background pixel or pixmap. The subwindow-mode effectively is **ClipByChildren**. If width is zero, it is replaced with the current width of the window minus x. If height is zero, it is replaced with the current height of the window minus y. If the window has a defined background tile, the rectangle clipped by any children is filled with this tile. If the window has background **None**, the contents of the window are not changed. In either case, if exposures is **True**, one or more **Expose** events are generated for regions of the rectangle that are either visible or are being retained in a backing store. If you specify a window whose class is **InputOnly**, a **BadMatch** error results.

XClearArea can generate **BadMatch**, **BadValue**, and **BadWindow** errors.

The **XClearWindow** function clears the entire area in the specified window and is equivalent to **XClearArea** (display, w, 0, 0, 0, 0, **False**). If the window has a defined background tile, the rectangle is tiled with a plane-mask of all ones and **GXcopy** function. If the window has background **None**, the contents of the window are not changed. If you specify a window whose class is **InputOnly**, a **BadMatch** error results.

XClearWindow can generate **BadMatch** and **BadWindow** errors.

DIAGNOSTICS

BadMatch	An **InputOnly** window is used as a Drawable.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XCopyArea(3)

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XClientMessageEvent – ColormapNotify event structure

STRUCTURES

The structure for **ClientMessage** events contains:

```
typedef struct {
        int type;                       /* ClientMessage */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        Atom message_type;
        int format;
        union {
                char b[20];
                short s[10];
                long l[5];
              } data;
} XClientMessageEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The message_type member is set to an atom that indicates how the data should be interpreted by the receiving client. The format member is set to 8, 16, or 32 and specifies whether the data should be viewed as a list of bytes, shorts, or longs. The data member is a union that contains the members b, s, and l. The b, s, and l members represent data of 20 8-bit values, 10 16-bit values, and 5 32-bit values. Particular message types might not make use of all these values. The X server places no interpretation on the values in the message_type or data members.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XColormapEvent – ColormapNotify event structure

STRUCTURES

The structure for **ColormapNotify** events contains:

```
typedef struct {
        int type;                       /* ColormapNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        Colormap colormap;              /* colormap or None */
        Bool new;
        int state;                      /* ColormapInstalled, ColormapUninstalled */
} XColormapEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window whose associated colormap is changed, installed, or uninstalled. For a colormap that is changed, installed, or uninstalled, the colormap member is set to the colormap associated with the window. For a colormap that is changed by a call to **XFreeColormap**, the colormap member is set to **None**. The new member is set to indicate whether the colormap for the specified window was changed or installed or uninstalled and can be **True** or **False**. If it is **True**, the colormap was changed. If it is **False**, the colormap was installed or uninstalled. The state member is always set to indicate whether the colormap is installed or uninstalled and can be **ColormapInstalled** or **ColormapUninstalled**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCreateColormap(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XConfigureEvent – ConfigureNotify event structure

STRUCTURES

The structure for **ConfigureNotify** events contains:

```
typedef struct {
        int type;                       /* ConfigureNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
        int x, y;
        int width, height;
        int border_width;
        Window above;
        Bool override_redirect;
} XConfigureEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the reconfigured window or to its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. The window member is set to the window whose size, position, border, and/or stacking order was changed.

The x and y members are set to the coordinates relative to the parent window's origin and indicate the position of the upper-left outside corner of the window. The width and height members are set to the inside size of the window, not including the border. The border_width member is set to the width of the window's border, in pixels.

The above member is set to the sibling window and is used for stacking operations. If the X server sets this member to **None**, the window whose state was changed is on the bottom of the stack with respect to sibling windows. However, if this member is set to a sibling window, the window whose state was changed is placed on top of this sibling window.

The override_redirect member is set to the override-redirect attribute of the window. Window manager clients normally should ignore this window if the override_redirect member is **True**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XConfigureRequestEvent – ConfigureRequest event structure

STRUCTURES

The structure for **ConfigureRequest** events contains:

```
typedef struct {
        int type;                       /* ConfigureRequest */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window parent;
        Window window;
        int x, y;
        int width, height;
        int border_width;
        Window above;
        int detail;                     /* Above, Below, TopIf, BottomIf, Opposite */
        unsigned long value_mask;
} XConfigureRequestEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The parent member is set to the parent window. The window member is set to the window whose size, position, border width, and/or stacking order is to be reconfigured. The value_mask member indicates which components were specified in the **ConfigureWindow** protocol request. The corresponding values are reported as given in the request. The remaining values are filled in from the current geometry of the window, except in the case of above (sibling) and detail (stack-mode), which are reported as **Above** and **None**, respectively, if they are not given in the request.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XConfigureWindow, XMoveWindow, XResizeWindow, XMoveResizeWindow, XSetWindowBorderWidth, XWindowChanges – configure windows and window changes structure

SYNTAX

XConfigureWindow(*display, w, value_mask, values*)
Display **display*;
Window *w*;
unsigned int *value_mask*;
XWindowChanges **values*;

XMoveWindow(*display, w, x, y*)
Display **display*;
Window *w*;
int *x, y*;

XResizeWindow(*display, w, width, height*)
Display **display*;
Window *w*;
unsigned int *width, height*;

XMoveResizeWindow(*display, w, x, y, width, height*)
Display **display*;
Window *w*;
int *x, y*;
unsigned int *width, height*;

XSetWindowBorderWidth(*display, w, width*)
Display **display*;
Window *w*;
unsigned int *width*;

ARGUMENTS

display	Specifies the connection to the X server.
value_mask	Specifies which values are to be set using information in the values structure. This mask is the bitwise inclusive OR of the valid configure window values bits.
values	Specifies a pointer to the **XWindowChanges** structure.
w	Specifies the window to be reconfigured, moved, or resized..
width	Specifies the width of the window border.
width *height*	Specify the width and height, which are the interior dimensions of the window.
x *y*	Specify the x and y coordinates, which define the new location of the top-left pixel of the window's border or the window itself if it has no border or define the new position of the window relative to its parent.

DESCRIPTION

The **XConfigureWindow** function uses the values specified in the **XWindowChanges** structure to reconfigure a window's size, position, border, and stacking order. Values not specified are taken from the existing geometry of the window.

If a sibling is specified without a stack_mode or if the window is not actually a sibling, a **BadMatch** error results. Note that the computations for **BottomIf**, **TopIf**, and **Opposite** are performed with respect to the window's final geometry (as controlled by the other arguments passed to **XConfigureWindow**), not its initial geometry. Any backing store contents of the window, its inferiors, and other newly visible windows are either discarded or changed to reflect the

current screen contents (depending on the implementation).

XConfigureWindow can generate **BadMatch**, **BadValue**, and **BadWindow** errors.

The **XMoveWindow** function moves the specified window to the specified x and y coordinates, but it does not change the window's size, raise the window, or change the mapping state of the window. Moving a mapped window may or may not lose the window's contents depending on if the window is obscured by nonchildren and if no backing store exists. If the contents of the window are lost, the X server generates **Expose** events. Moving a mapped window generates **Expose** events on any formerly obscured windows.

If the override-redirect flag of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no further processing is performed. Otherwise, the window is moved.

XMoveWindow can generate a **BadWindow** error.

The **XResizeWindow** function changes the inside dimensions of the specified window, not including its borders. This function does not change the window's upper-left coordinate or the origin and does not restack the window. Changing the size of a mapped window may lose its contents and generate **Expose** events. If a mapped window is made smaller, changing its size generates **Expose** events on windows that the mapped window formerly obscured.

If the override-redirect flag of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no further processing is performed. If either width or height is zero, a **BadValue** error results.

XResizeWindow can generate **BadValue** and **BadWindow** errors.

The **XMoveResizeWindow** function changes the size and location of the specified window without raising it. Moving and resizing a mapped window may generate an **Expose** event on the window. Depending on the new size and location parameters, moving and resizing a window may generate **Expose** events on windows that the window formerly obscured.

If the override-redirect flag of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no further processing is performed. Otherwise, the window size and location are changed.

XMoveResizeWindow can generate **BadValue** and **BadWindow** errors.

The **XSetWindowBorderWidth** function sets the specified window's border width to the specified width.

XSetWindowBorderWidth can generate a **BadWindow** error.

STRUCTURES

The **XWindowChanges** structure contains:

```
/* Configure window value mask bits */
#define     CWX                 (1<<0)
#define     CWY                 (1<<1)
#define     CWWidth             (1<<2)
#define     CWHeight            (1<<3)
#define     CWBorderWidth       (1<<4)
#define     CWSibling           (1<<5)
#define     CWStackMode         (1<<6)
/* Values */

typedef struct {
        int x, y;
        int width, height;
        int border_width;
```

```
        Window sibling;
        int stack_mode;
} XWindowChanges;
```

The x and y members are used to set the window's x and y coordinates, which are relative to the parent's origin and indicate the position of the upper-left outer corner of the window. The width and height members are used to set the inside size of the window, not including the border, and must be nonzero, or a **BadValue** error results. Attempts to configure a root window have no effect.

The border_width member is used to set the width of the border in pixels. Note that setting just the border width leaves the outer-left corner of the window in a fixed position but moves the absolute position of the window's origin. If you attempt to set the border-width attribute of an **InputOnly** window nonzero, a **BadMatch** error results.

The sibling member is used to set the sibling window for stacking operations. The stack_mode member is used to set how the window is to be restacked and can be set to **Above**, **Below**, **TopIf**, **BottomIf**, or **Opposite**.

DIAGNOSTICS

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XCreateWindow(3), XDestroyWindow(3), XMapWindow(3), XRaiseWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCopyArea, XCopyPlane – copy areas

SYNTAX

XCopyArea(*display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y*)
Display **display*;
Drawable *src, dest*;
GC *gc*;
int *src_x, src_y*;
unsigned int *width, height*;
int *dest_x, dest_y*;

XCopyPlane(*display, src, dest, gc, src_x, src_y, width, height, dest_x, dest_y, plane*)
Display **display*;
Drawable *src, dest*;
GC *gc*;
int *src_x, src_y*;
unsigned int *width, height*;
int *dest_x, dest_y*;
unsigned long *plane*;

ARGUMENTS

dest_x
dest_y — Specify the x and y coordinates, which are relative to the origin of the destination rectangle and specify its upper-left corner.

display — Specifies the connection to the X server.

gc — Specifies the GC.

plane — Specifies the bit plane. You must set exactly one bit to 1.

src
dest — Specify the source and destination rectangles to be combined.

src_x
src_y — Specify the x and y coordinates, which are relative to the origin of the source rectangle and specify its upper-left corner.

width
height — Specify the width and height, which are the dimensions of both the source and destination rectangles.

DESCRIPTION

The **XCopyArea** function combines the specified rectangle of src with the specified rectangle of dest. The drawables must have the same root and depth, or a **BadMatch** error results.

If regions of the source rectangle are obscured and have not been retained in backing store or if regions outside the boundaries of the source drawable are specified, those regions are not copied. Instead, the following occurs on all corresponding destination regions that are either visible or are retained in backing store. If the destination is a window with a background other than **None**, corresponding regions of the destination are tiled with that background (with plane-mask of all ones and **GXcopy** function). Regardless of tiling or whether the destination is a window or a pixmap, if graphics-exposures is **True**, then **GraphicsExpose** events for all corresponding destination regions are generated. If graphics-exposures is **True** but no **GraphicsExpose** events are generated, a **NoExpose** event is generated. Note that by default graphics-exposures is **True** in new GCs.

This function uses these GC components: function, plane-mask, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.

XCopyArea can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

The **XCopyPlane** function uses a single bit plane of the specified source rectangle combined with the specified GC to modify the specified rectangle of dest. The drawables must have the same root but need not have the same depth. If the drawables do not have the same root, a **BadMatch** error results. If plane does not have exactly one bit set to 1 and the values of planes must be less than %2 sup n%, where *n* is the depth of scr, a **BadValue** error results.

Effectively, **XCopyPlane** forms a pixmap of the same depth as the rectangle of dest and with a size specified by the source region. It uses the foreground/background pixels in the GC (foreground everywhere the bit plane in src contains a bit set to 1, background everywhere the bit plane in src contains a bit set to 0) and the equivalent of a **CopyArea** protocol request is performed with all the same exposure semantics. This can also be thought of as using the specified region of the source bit plane as a stipple with a fill-style of **FillOpaqueStippled** for filling a rectangular area of the destination.

This function uses these GC components: function, plane-mask, foreground, background, subwindow-mode, graphics-exposures, clip-x-origin, clip-y-origin, and clip-mask.

XCopyPlane can generate **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue** errors.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XClearArea(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateColormap, XCopyColormapAndFree, XFreeColormap, XSetWindowColormap, XColor – create, copy, or destroy colormaps and color structure

SYNTAX

Colormap XCreateColormap(*display, w, visual, alloc*)
Display **display*;
Window *w*;
Visual **visual*;
int *alloc*;

Colormap XCopyColormapAndFree(*display, colormap*)
Display **display*;
Colormap *colormap*;

XFreeColormap(*display, colormap*)
Display **display*;
Colormap *colormap*;

XSetWindowColormap(*display, w, colormap*)
Display **display*;
Window *w*;
Colormap *colormap*;

ARGUMENTS

alloc	Specifies the colormap entries to be allocated. You can pass **AllocNone** or **AllocAll**.
colormap	Specifies the colormap that you want to create, copy, set, or destroy.
display	Specifies the connection to the X server.
visual	Specifies a pointer to a visual type supported on the screen. If the visual type is not one supported by the screen, a **BadMatch** error results.
w	Specifies the window for which you want to create or set a colormap .

DESCRIPTION

The **XCreateColormap** function creates a colormap of the specified visual type for the screen on which the specified window resides and returns the colormap ID associated with it. Note that the specified window is only used to determine the screen.

The initial values of the colormap entries are undefined for the visual classes **GrayScale**, **PseudoColor**, and **DirectColor**. For **StaticGray**, **StaticColor**, and **TrueColor**, the entries have defined values, but those values are specific to the visual and are not defined by X. For **StaticGray**, **StaticColor**, and **TrueColor**, alloc must be **AllocNone**, or a **BadMatch** error results. For the other visual classes, if alloc is **AllocNone**, the colormap initially has no allocated entries, and clients can allocate them. For information about the visual types, see section 3.1.

If alloc is **AllocAll**, the entire colormap is allocated writable. The initial values of all allocated entries are undefined. For **GrayScale** and **PseudoColor**, the effect is as if an **XAllocColorCells** call returned all pixel values from zero to N – 1, where N is the colormap entries value in the specified visual. For **DirectColor**, the effect is as if an **XAllocColorPlanes** call returned a pixel value of zero and red_mask, green_mask, and blue_mask values containing the same bits as the corresponding masks in the specified visual. However, in all cases, none of these entries can be freed by using **XFreeColors**.

XCreateColormap can generate **BadAlloc**, **BadMatch**, **BadValue**, and **BadWindow** errors.

The **XCopyColormapAndFree** function creates a colormap of the same visual type and for the same screen as the specified colormap and returns the new colormap ID. It also moves all of the client's existing allocation from the specified colormap to the new colormap with their color values intact and their read-only or writable characteristics intact and frees those entries in the specified colormap. Color values in other entries in the new colormap are undefined. If the specified colormap was created by the client with alloc set to **AllocAll**, the new colormap is also created with **AllocAll**, all color values for all entries are copied from the specified colormap, and then all entries in the specified colormap are freed. If the specified colormap was not created by the client with **AllocAll**, the allocations to be moved are all those pixels and planes that have been allocated by the client using **XAllocColor**, **XAllocNamedColor**, **XAllocColorCells**, or **XAllocColorPlanes** and that have not been freed since they were allocated.

XCopyColormapAndFree can generate **BadAlloc** and **BadColor** errors.

The **XFreeColormap** function deletes the association between the colormap resource ID and the colormap and frees the colormap storage. However, this function has no effect on the default colormap for a screen. If the specified colormap is an installed map for a screen, it is uninstalled (see **XUninstallColormap**). If the specified colormap is defined as the colormap for a window (by **XCreateWindow**, **XSetWindowColormap**, or **XChangeWindowAttributes**), **XFreeColormap** changes the colormap associated with the window to **None** and generates a **ColormapNotify** event. X does not define the colors displayed for a window with a colormap of **None**.

XFreeColormap can generate a **BadColor** error.

The **XSetWindowColormap** function sets the specified colormap of the specified window. The colormap must have the same visual type as the window, or a **BadMatch** error results.

XSetWindowColormap can generate **BadColor**, **BadMatch**, and **BadWindow** errors.

STRUCTURES

The **XColor** structure contains:

```
typedef struct {
        unsigned long pixel;              /* pixel value */
        unsigned short red, green, blue; /* rgb values */
        char flags;                       /* DoRed, DoGreen, DoBlue */
        char pad;
} XColor;
```

The red, green, and blue values are scaled between 0 and 65535. On full in a color is a value of 65535 independent of the number of bits actually used in the display hardware. Half brightness in a color is a value of 32767, and off is 0. This representation gives uniform results for color values across different screens. In some functions, the flags member controls which of the red, green, and blue members is used and can be one or more of **DoRed**, **DoGreen**, and **DoBlue**.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadColor	A value for a Colormap argument does not name a defined Colormap.
BadMatch	An **InputOnly** window is used as a Drawable.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocColor(3), XChangeWindowAtrributes(3), XCreateWindow(3), XQueryColor(3), XStoreColors(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateFontCursor, XCreatePixmapCursor, XCreateGlyphCursor - create cursors

SYNTAX

#include <X11/cursorfont.h>

Cursor XCreateFontCursor(*display, shape*)
Display **display*;
unsigned int *shape*;

Cursor XCreatePixmapCursor(*display, source, mask, foreground_color, background_color, x, y*)
Display **display*;
Pixmap *source*;
Pixmap *mask*;
XColor **foreground_color*;
XColor **background_color*;
unsigned int *x, y*;

Cursor XCreateGlyphCursor(*display, source_font, mask_font, source_char, mask_char, foreground_color, background_color*)
Display **display*;
Font *source_font, mask_font*;
unsigned int *source_char, mask_char*;
XColor **foreground_color*;
XColor **background_color*;

ARGUMENTS

background_color
Specifies the RGB values for the background of the source.

display Specifies the connection to the X server.

foreground_color
Specifies the RGB values for the foreground of the source.

mask Specifies the cursor's source bits to be displayed or **None**.

mask_char Specifies the glyph character for the mask.

mask_font Specifies the font for the mask glyph or **None**.

shape Specifies the shape of the cursor.

source Specifies the shape of the source cursor.

source_char Specifies the character glyph for the source.

source_font Specifies the font for the source glyph.

x
y Specify the x and y coordinates, which indicate the hotspot relative to the source's origin.

DESCRIPTION

X provides a set of standard cursor shapes in a special font named cursor. Applications are encouraged to use this interface for their cursors because the font can be customized for the individual display type. The shape argument specifies which glyph of the standard fonts to use.

The hotspot comes from the information stored in the cursor font. The initial colors of a cursor are a black foreground and a white background (see **XRecolorCursor**).

XCreateFontCursor can generate **BadAlloc** and **BadValue** errors.

The **XCreatePixmapCursor** function creates a cursor and returns the cursor ID associated with it. The foreground and background RGB values must be specified using foreground_color and background_color, even if the X server only has a **StaticGray** or **GrayScale** screen. The foreground color is used for the pixels set to 1 in the source, and the background color is used for the pixels set to 0. Both source and mask, if specified, must have depth one (or a **BadMatch** error results) but can have any root. The mask argument defines the shape of the cursor. The pixels set to 1 in the mask define which source pixels are displayed, and the pixels set to 0 define which pixels are ignored. If no mask is given, all pixels of the source are displayed. The mask, if present, must be the same size as the pixmap defined by the source argument, or a **BadMatch** error results. The hotspot must be a point within the source, or a **BadMatch** error results.

The components of the cursor can be transformed arbitrarily to meet display limitations. The pixmaps can be freed immediately if no further explicit references to them are to be made. Subsequent drawing in the source or mask pixmap has an undefined effect on the cursor. The X server might or might not make a copy of the pixmap.

XCreatePixmapCursor can generate **BadAlloc** and **BadPixmap** errors.

The **XCreateGlyphCursor** function is similar to **XCreatePixmapCursor** except that the source and mask bitmaps are obtained from the specified font glyphs. The source_char must be a defined glyph in source_font, or a **BadValue** error results. If mask_font is given, mask_char must be a defined glyph in mask_font, or a **BadValue** error results. The mask_font and character are optional. The origins of the source_char and mask_char (if defined) glyphs are positioned coincidently and define the hotspot. The source_char and mask_char need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes. If no mask_char is given, all pixels of the source are displayed. You can free the fonts immediately by calling **XFreeFont** if no further explicit references to them are to be made.

For 2-byte matrix fonts, the 16-bit value should be formed with the byte1 member in the most-significant byte and the byte2 member in the least-significant byte.

XCreateGlyphCursor can generate **BadAlloc**, **BadFont**, and **BadValue** errors.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadFont A value for a Font or GContext argument does not name a defined Font.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadPixmap A value for a Pixmap argument does not name a defined Pixmap.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XDefineCursor(3), XLoadFont(3), XRecolorCursor(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateGC, XCopyGC, XChangeGC, XGetGCValues, XFreeGC, XGContextFromGC, XGCValues – create or free graphics contexts and graphics context structure

SYNTAX

GC XCreateGC(*display, d, valuemask, values*)
Display **display*;
Drawable *d*;
unsigned long *valuemask*;
XGCValues **values*;

XCopyGC(*display, src, valuemask, dest*)
Display **display*;
GC *src, dest*;
unsigned long *valuemask*;

XChangeGC(*display, gc, valuemask, values*)
Display **display*;
GC *gc*;
unsigned long *valuemask*;
XGCValues **values*;

Status XGetGCValues(*display, gc, valuemask, values_return*)
Display **display*;
GC *gc*;
unsigned long *valuemask*;
XGCValues **values_return*;

XFreeGC(*display, gc*)
Display **display*;
GC *gc*;

GContext XGContextFromGC(*gc*)
GC *gc*;

ARGUMENTS

d	Specifies the drawable.
dest	Specifies the destination GC.
display	Specifies the connection to the X server.
gc	Specifies the GC.
src	Specifies the components of the source GC.
valuemask	Specifies which components in the GC are to be set, copied, changed, or returned . This argument is the bitwise inclusive OR of one or more of the valid GC component mask bits.
values	Specifies any values as specified by the valuemask.
values_return	Returns the GC values in the specified **XGCValues** structure.

DESCRIPTION

The **XCreateGC** function creates a graphics context and returns a GC. The GC can be used with any destination drawable having the same root and depth as the specified drawable. Use with other drawables results in a **BadMatch** error.

XCreateGC can generate **BadAlloc**, **BadDrawable**, **BadFont**, **BadMatch**, **BadPixmap**, and **BadValue** errors.

The **XCopyGC** function copies the specified components from the source GC to the destination GC. The source and destination GCs must have the same root and depth, or a **BadMatch** error results. The valuemask specifies which component to copy, as for **XCreateGC**.

XCopyGC can generate **BadAlloc**, **BadGC**, and **BadMatch** errors.

The **XChangeGC** function changes the components specified by valuemask for the specified GC. The values argument contains the values to be set. The values and restrictions are the same as for **XCreateGC**. Changing the clip-mask overrides any previous **XSetClipRectangles** request on the context. Changing the dash-offset or dash-list overrides any previous **XSetDashes** request on the context. The order in which components are verified and altered is server-dependent. If an error is generated, a subset of the components may have been altered.

XChangeGC can generate **BadAlloc**, **BadFont**, **BadGC**, **BadMatch**, **BadPixmap**, and **BadValue** errors.

The **XGetGCValues** function returns the components specified by valuemask for the specified GC. Note that the clip mask and dash list (represented by the **GCClipMask** and **GCDashList** bits, respectively, in the valuemask) cannot be requested. If the valuemask contains a valid set of GC mask bits (**GCFunction**, **GCPlaneMask**, **GCForeground**, **GCBackground**, **GCLineWidth**, **GCLineStyle**, **GCCapStyle**, **GCJoinStyle**, **GCFillStyle**, **GCFillRule**, **GCTile**, **GCStipple**, **GCTileStipXOrigin**, **GCTileStipYOrigin**, **GCFont**, **GCSubwindowMode**, **GCGraphicsExposures**, **GCClipXOrigin**, **GCClipYOrigin**, **GCDashOffset**, or **GCArcMode**) and no error occur, **XGetGCValues** sets the requested components in values_return and returns a nonzero status. Otherwise, it returns a zero status.

The **XFreeGC** function destroys the specified GC as well as all the associated storage.

XFreeGC can generate a **BadGC** error.

STRUCTURES

The **XGCValues** structure contains:

```
/* GC attribute value mask bits */
#define  GCFunction           (1L<<0)
#define  GCPlaneMask          (1L<<1)
#define  GCForeground         (1L<<2)
#define  GCBackground         (1L<<3)
#define  GCLineWidth          (1L<<4)
#define  GCLineStyle          (1L<<5)
#define  GCCapStyle           (1L<<6)
#define  GCJoinStyle          (1L<<7)
#define  GCFillStyle          (1L<<8)
#define  GCFillRule           (1L<<9)
#define  GCTile               (1L<<10)
#define  GCStipple            (1L<<11)
#define  GCTileStipXOrigin    (1L<<12)
#define  GCTileStipYOrigin    (1L<<13)
#define  GCFont               (1L<<14)
#define  GCSubwindowMode      (1L<<15)
#define  GCGraphicsExposures  (1L<<16)
#define  GCClipXOrigin        (1L<<17)
#define  GCClipYOrigin        (1L<<18)
#define  GCClipMask           (1L<<19)
#define  GCDashOffset         (1L<<20)
#define  GCDashList           (1L<<21)
#define  GCArcMode            (1L<<22)
/* Values */
```

```
typedef struct {
        int function;                   /* logical operation */
        unsigned long plane_mask;       /* plane mask */
        unsigned long foreground;       /* foreground pixel */
        unsigned long background;       /* background pixel */
        int line_width;                 /* line width (in pixels) */
        int line_style;                 /* LineSolid, LineOnOffDash, LineDoubleDash */
        int cap_style;                  /* CapNotLast, CapButt, CapRound, CapProjecting */
        int join_style;                 /* JoinMiter, JoinRound, JoinBevel */
        int fill_style;                 /* FillSolid, FillTiled, FillStippled FillOpaqueStippled*/
        int fill_rule;                  /* EvenOddRule, WindingRule */
        int arc_mode;                   /* ArcChord, ArcPieSlice */
        Pixmap tile;                    /* tile pixmap for tiling operations */
        Pixmap stipple;                 /* stipple 1 plane pixmap for stippling */
        int ts_x_origin;                /* offset for tile or stipple operations */
        int ts_y_origin;
        Font font;                      /* default text font for text operations */
        int subwindow_mode;             /* ClipByChildren, IncludeInferiors */
        Bool graphics_exposures;        /* boolean, should exposures be generated */
        int clip_x_origin;              /* origin for clipping */
        int clip_y_origin;
        Pixmap clip_mask;               /* bitmap clipping; other calls for rects */
        int dash_offset;                /* patterned/dashed line information */
        char dashes;
} XGCValues;
```

The function attributes of a GC are used when you update a section of a drawable (the destination) with bits from somewhere else (the source). The function in a GC defines how the new destination bits are to be computed from the source bits and the old destination bits. **GXcopy** is typically the most useful because it will work on a color display, but special applications may use other functions, particularly in concert with particular planes of a color display. The 16 GC functions, defined in **<X11/X.h>**, are:

Function Name	Hex Code	Operation
.ne 50u+1u		
GXclear	0x0	0
GXand	0x1	src AND dst
GXandReverse	0x2	src AND NOT dst
GXcopy	0x3	src
GXandInverted	0x4	(NOT src) AND dst
GXnoop	0x5	dst
GXxor	0x6	src XOR dst
GXor	0x7	src OR dst
GXnor	0x8	(NOT src) AND (NOT dst)
GXequiv	0x9	(NOT src) XOR dst
GXinvert	0xa	NOT dst
GXorReverse	0xb	src OR (NOT dst)
GXcopyInverted	0xc	NOT src
GXorInverted	0xd	(NOT src) OR dst
GXnand	0xe	(NOT src) OR (NOT dst)
GXset	0xf	1

Many graphics operations depend on either pixel values or planes in a GC. The planes attribute is of type long, and it specifies which planes of the destination are to be modified, one bit per plane. A monochrome display has only one plane and will be the least-significant bit of the word. As planes are added to the display hardware, they will occupy more significant bits in the plane mask.

In graphics operations, given a source and destination pixel, the result is computed bitwise on corresponding bits of the pixels. That is, a Boolean operation is performed in each bit plane. The plane_mask restricts the operation to a subset of planes. A macro constant **AllPlanes** can be used to refer to all planes of the screen simultaneously. The result is computed by the following:

*((src FUNC dst) AND plane-mask) OR (dst AND (NOT plane-mask))

Range checking is not performed on the values for foreground, background, or plane_mask. They are simply truncated to the appropriate number of bits. The line-width is measured in pixels and either can be greater than or equal to one (wide line) or can be the special value zero (thin line).

Wide lines are drawn centered on the path described by the graphics request. Unless otherwise specified by the join-style or cap-style, the bounding box of a wide line with endpoints [x1, y1], [x2, y2] and width w is a rectangle with vertices at the following real coordinates:

[x1-(w*sn/2), y1+(w*cs/2)], [x1+(w*sn/2), y1-(w*cs/2)],
[x2-(w*sn/2), y2+(w*cs/2)], [x2+(w*sn/2), y2-(w*cs/2)]

Here sn is the sine of the angle of the line, and cs is the cosine of the angle of the line. A pixel is part of the line and so is drawn if the center of the pixel is fully inside the bounding box (which is viewed as having infinitely thin edges). If the center of the pixel is exactly on the bounding box, it is part of the line if and only if the interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are part of the line if and only if the interior or the boundary is immediately below (y increasing direction) and the interior or the boundary is immediately to the right (x increasing direction).

Thin lines (zero line-width) are one-pixel-wide lines drawn using an unspecified, device-dependent algorithm. There are only two constraints on this algorithm.

1. If a line is drawn unclipped from [x1,y1] to [x2,y2] and if another line is drawn unclipped from [x1+dx,y1+dy] to [x2+dx,y2+dy], a point [x,y] is touched by drawing the first line if and only if the point [x+dx,y+dy] is touched by drawing the second line.
2. The effective set of points comprising a line cannot be affected by clipping. That is, a point is touched in a clipped line if and only if the point lies inside the clipping region and the point would be touched by the line when drawn unclipped.

A wide line drawn from [x1,y1] to [x2,y2] always draws the same pixels as a wide line drawn from [x2,y2] to [x1,y1], not counting cap-style and join-style. It is recommended that this property be true for thin lines, but this is not required. A line-width of zero may differ from a line-width of one in which pixels are drawn. This permits the use of many manufacturers' line drawing hardware, which may run many times faster than the more precisely specified wide lines.

In general, drawing a thin line will be faster than drawing a wide line of width one. However, because of their different drawing algorithms, thin lines may not mix well aesthetically with wide lines. If it is desirable to obtain precise and uniform results across all displays, a client should always use a line-width of one rather than a line-width of zero.

The line-style defines which sections of a line are drawn:

LineSolid The full path of the line is drawn.

LineDoubleDash The full path of the line is drawn, but the even dashes are filled differently than the odd dashes (see fill-style) with **CapButt** style used where even and odd dashes meet.

LineOnOffDash Only the even dashes are drawn, and cap-style applies to all internal ends of the individual dashes, except **CapNotLast** is treated as **CapButt**.

The cap-style defines how the endpoints of a path are drawn:

CapNotLast This is equivalent to **CapButt** except that for a line-width of zero the final endpoint is not drawn.

CapButt The line is square at the endpoint (perpendicular to the slope of the line) with no projection beyond.

CapRound The line has a circular arc with the diameter equal to the line-width, centered on the endpoint. (This is equivalent to **CapButt** for line-width of zero).

CapProjecting The line is square at the end, but the path continues beyond the endpoint for a distance equal to half the line-width. (This is equivalent to **CapButt** for line-width of zero).

The join-style defines how corners are drawn for wide lines:

JoinMiter The outer edges of two lines extend to meet at an angle. However, if the angle is less than 11 degrees, then a **JoinBevel** join-style is used instead.

JoinRound The corner is a circular arc with the diameter equal to the line-width, centered on the joinpoint.

JoinBevel The corner has **CapButt** endpoint styles with the triangular notch filled.

For a line with coincident endpoints (x1=x2, y1=y2), when the cap-style is applied to both endpoints, the semantics depends on the line-width and the cap-style:

CapNotLast	thin	The results are device-dependent, but the desired effect is that nothing is drawn.
CapButt	thin	The results are device-dependent, but the desired effect is that a single pixel is drawn.
CapRound	thin	The results are the same as for **CapButt**/thin.
CapProjecting	thin	The results are the same as for **Butt**/thin.
CapButt	wide	Nothing is drawn.
CapRound	wide	The closed path is a circle, centered at the endpoint, and with the diameter equal to the line-width.
CapProjecting	wide	The closed path is a square, aligned with the coordinate axes, centered at the endpoint, and with the sides equal to the line-width.

For a line with coincident endpoints (x1=x2, y1=y2), when the join-style is applied at one or both endpoints, the effect is as if the line was removed from the overall path. However, if the total path consists of or is reduced to a single point joined with itself, the effect is the same as when the cap-style is applied at both endpoints.

The tile/stipple and clip origins are interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The tile pixmap must have the same root and depth as the GC, or a **BadMatch** error results. The stipple pixmap must have depth one and must have the same root as the GC, or a **BadMatch** error results. For stipple operations where the fill-style is **FillStippled** but not **FillOpaqueStippled**, the stipple pattern is tiled in a single plane and acts as an additional clip mask to be ANDed with the clip-mask. Although some sizes may be faster to use than others, any size pixmap can be used for tiling or stippling.

The fill-style defines the contents of the source for line, text, and fill requests. For all text and fill requests (for example, **XDrawText**, **XDrawText16**, **XFillRectangle**, **XFillPolygon**, and **XFillArc**); for line requests with line-style **LineSolid** (for example, **XDrawLine**, **XDrawSegments**, **XDrawRectangle**, **XDrawArc**); and for the even dashes for line requests with line-style **LineOnOffDash** or **LineDoubleDash**, the following apply:

FillSolid	Foreground
FillTiled	Tile
FillOpaqueStippled	A tile with the same width and height as stipple, but with background everywhere stipple has a zero and with foreground everywhere stipple has a one
FillStippled	Foreground masked by stipple

When drawing lines with line-style **LineDoubleDash**, the odd dashes are controlled by the fill-style in the following manner:

FillSolid	Background
FillTiled	Same as for even dashes
FillOpaqueStippled	Same as for even dashes
FillStippled	Background masked by stipple

Storing a pixmap in a GC might or might not result in a copy being made. If the pixmap is later used as the destination for a graphics request, the change might or might not be reflected in the GC. If the pixmap is used simultaneously in a graphics request both as a destination and as a tile or stipple, the results are undefined.

For optimum performance, you should draw as much as possible with the same GC (without changing its components). The costs of changing GC components relative to using different GCs depend upon the display hardware and the server implementation. It is quite likely that some amount of GC information will be cached in display hardware and that such hardware can only cache a small number of GCs.

The dashes value is actually a simplified form of the more general patterns that can be set with **XSetDashes**. Specifying a value of N is equivalent to specifying the two-element list [N, N] in **XSetDashes**. The value must be nonzero, or a **BadValue** error results.

The clip-mask restricts writes to the destination drawable. If the clip-mask is set to a pixmap, it must have depth one and have the same root as the GC, or a **BadMatch** error results. If clip-mask is set to **None**, the pixels are always drawn regardless of the clip origin. The clip-mask also can be set by calling the **XSetClipRectangles** or **XSetRegion** functions. Only pixels where the clip-mask has a bit set to 1 are drawn. Pixels are not drawn outside the area covered by the clip-mask or where the clip-mask has a bit set to 0. The clip-mask affects all graphics requests. The clip-mask does not clip sources. The clip-mask origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request.

You can set the subwindow-mode to **ClipByChildren** or **IncludeInferiors**. For **ClipByChildren**, both source and destination windows are additionally clipped by all viewable **InputOutput** children. For **IncludeInferiors**, neither source nor destination window is clipped by inferiors. This will result in including subwindow contents in the source and drawing through subwindow boundaries of the destination. The use of **IncludeInferiors** on a window of one depth with mapped inferiors of differing depth is not illegal, but the semantics are undefined by the core protocol.

The fill-rule defines what pixels are inside (drawn) for paths given in **XFillPolygon** requests and can be set to **EvenOddRule** or **WindingRule**. For **EvenOddRule**, a point is inside if an infinite ray with the point as origin crosses the path an odd number of times. For **WindingRule**, a point is inside if an infinite ray with the point as origin crosses an unequal number of clockwise

and counterclockwise directed path segments. A clockwise directed path segment is one that crosses the ray from left to right as observed from the point. A counterclockwise segment is one that crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because you can simply choose a different ray that is not coincident with a segment.

For both **EvenOddRule** and **WindingRule**, a point is infinitely small, and the path is an infinitely thin line. A pixel is inside if the center point of the pixel is inside and the center point is not on the boundary. If the center point is on the boundary, the pixel is inside if and only if the polygon interior is immediately to its right (x increasing direction). Pixels with centers on a horizontal edge are a special case and are inside if and only if the polygon interior is immediately below (y increasing direction).

The arc-mode controls filling in the **XFillArcs** function and can be set to **ArcPieSlice** or **ArcChord**. For **ArcPieSlice**, the arcs are pie-slice filled. For **ArcChord**, the arcs are chord filled.

The graphics-exposure flag controls **GraphicsExpose** event generation for **XCopyArea** and **XCopyPlane** requests (and any similar requests defined by extensions).

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadFont A value for a Font or GContext argument does not name a defined Font.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadPixmap A value for a Pixmap argument does not name a defined Pixmap.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

AllPlanes(3), XCopyArea(3), XCreateRegion(3), XDrawArc(3), XDrawLine(3), XDrawRectangle(3), XDrawText(3), XFillRectangle(3), XQueryBestSize(3), XSetArcMode(3), XSetClipOrigin(3), XSetFillStyle(3), XSetFont(3), XSetLineAttributes(3), XSetState(3), XSetTile(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateImage, XGetPixel, XPutPixel, XSubImage, XAddPixel, XDestroyImage – image utilities

SYNTAX

XImage *XCreateImage(*display, visual, depth, format, offset, data, width, height, bitmap_pad, bytes_per_line*)
Display **display*;
Visual **visual*;
unsigned int *depth*;
int *format*;
int *offset*;
char **data*;
unsigned int *width*;
unsigned int *height*;
int *bitmap_pad*;
int *bytes_per_line*;

unsigned long XGetPixel(*ximage, x, y*)
XImage **ximage*;
int *x*;
int *y*;

XPutPixel(*ximage, x, y, pixel*)
XImage **ximage*;
int *x*;
int *y*;
unsigned long *pixel*;

XImage *XSubImage(*ximage, x, y, subimage_width, subimage_height*)
XImage **ximage*;
int *x*;
int *y*;
unsigned int *subimage_width*;
unsigned int *subimage_height*;

XAddPixel(*ximage, value*)
XImage **ximage*;
long *value*;

XDestroyImage(*ximage*)
XImage **ximage*;

ARGUMENTS

bitmap_pad	Specifies the quantum of a scanline (8, 16, or 32). In other words, the start of one scanline is separated in client memory from the start of the next scanline by an integer multiple of this many bits.
bytes_per_line	Specifies the number of bytes in the client image between the start of one scanline and the start of the next.
data	Specifies a pointer to the image data.
depth	Specifies the depth of the image.
display	Specifies the connection to the X server.
format	Specifies the format for the image. You can pass **XYBitmap**, **XYPixmap**, or **ZPixmap**.
height	Specifies the height of the image, in pixels.
offset	Specifies the number of pixels to ignore at the beginning of the scanline.

pixel Specifies the new pixel value.

subimage_height
Specifies the height of the new subimage, in pixels.

subimage_width Specifies the width of the new subimage, in pixels.

value Specifies the constant value that is to be added.

visual Specifies a pointer to the visual.

width Specifies the width of the image, in pixels.

ximage Specifies a pointer to the image.

x
y Specify the x and y coordinates.

DESCRIPTION

The **XCreateImage** function allocates the memory needed for an **XImage** structure for the specified display but does not allocate space for the image itself. Rather, it initializes the structure byte-order, bit-order, and bitmap-unit values from the display and returns a pointer to the **XImage** structure. The red, green, and blue mask values are defined for Z format images only and are derived from the **Visual** structure passed in. Other values also are passed in. The offset permits the rapid displaying of the image without requiring each scanline to be shifted into position. If you pass a zero value in bytes_per_line, Xlib assumes that the scanlines are contiguous in memory and calculates the value of bytes_per_line itself.

Note that when the image is created using **XCreateImage**, **XGetImage**, or **XSubImage**, the destroy procedure that the **XDestroyImage** function calls frees both the image structure and the data pointed to by the image structure.

The basic functions used to get a pixel, set a pixel, create a subimage, and add a constant offset to a Z format image are defined in the image object. The functions in this section are really macro invocations of the functions in the image object and are defined in **<X11/Xutil.h>**.

The **XGetPixel** function returns the specified pixel from the named image. The pixel value is returned in normalized format (that is, the least-significant byte of the long is the least-significant byte of the pixel). The image must contain the x and y coordinates.

The **XPutPixel** function overwrites the pixel in the named image with the specified pixel value. The input pixel value must be in normalized format (that is, the least-significant byte of the long is the least-significant byte of the pixel). The image must contain the x and y coordinates.

The **XSubImage** function creates a new image that is a subsection of an existing one. It allocates the memory necessary for the new **XImage** structure and returns a pointer to the new image. The data is copied from the source image, and the image must contain the rectangle defined by x, y, subimage_width, and subimage_height.

The **XAddPixel** function adds a constant value to every pixel in an image. It is useful when you have a base pixel value from allocating color resources and need to manipulate the image to that form.

The **XDestroyImage** function deallocates the memory associated with the **XImage** structure.

SEE ALSO

XPutImage(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreatePixmap, XFreePixmap – create or destroy pixmaps

SYNTAX

Pixmap XCreatePixmap(*display, d, width, height, depth*)
Display **display*;
Drawable *d*;
unsigned int *width, height*;
unsigned int *depth*;

XFreePixmap(*display, pixmap*)
Display **display*;
Pixmap *pixmap*;

ARGUMENTS

d	Specifies which screen the pixmap is created on.
depth	Specifies the depth of the pixmap.
display	Specifies the connection to the X server.
pixmap	Specifies the pixmap.
width *height*	Specify the width and height, which define the dimensions of the pixmap.

DESCRIPTION

The **XCreatePixmap** function creates a pixmap of the width, height, and depth you specified and returns a pixmap ID that identifies it. It is valid to pass an **InputOnly** window to the drawable argument. The width and height arguments must be nonzero, or a **BadValue** error results. The depth argument must be one of the depths supported by the screen of the specified drawable, or a **BadValue** error results.

The server uses the specified drawable to determine on which screen to create the pixmap. The pixmap can be used only on this screen and only with other drawables of the same depth (see **XCopyPlane** for an exception to this rule). The initial contents of the pixmap are undefined.

XCreatePixmap can generate **BadAlloc**, **BadDrawable**, and **BadValue** errors.

The **XFreePixmap** function first deletes the association between the pixmap ID and the pixmap. Then, the X server frees the pixmap storage when there are no references to it. The pixmap should never be referenced again.

XFreePixmap can generate a **BadPixmap** error.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCopyArea(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateRegion, XSetRegion, XDestroyRegion – create or destroy regions

SYNTAX

Region XCreateRegion()

XSetRegion(*display*, *gc*, *r*)
Display **display*;
GC *gc*;
Region *r*;

XDestroyRegion(*r*)
Region *r*;

ARGUMENTS

display	Specifies the connection to the X server.
gc	Specifies the GC.
r	Specifies the region.

DESCRIPTION

The **XCreateRegion** function creates a new empty region.

The **XSetRegion** function sets the clip-mask in the GC to the specified region. Once it is set in the GC, the region can be destroyed.

The **XDestroyRegion** function deallocates the storage associated with a specified region.

SEE ALSO

XEmptyRegion(3), XIntersectRegion(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateWindow, XCreateSimpleWindow, XSetWindowAttributes – create windows and window attributes structure

SYNTAX

Window XCreateWindow(*display, parent, x, y, width, height, border_width, depth, class, visual, valuemask, attributes*)

Display **display*;
Window *parent*;
int *x, y*;
unsigned int *width, height*;
unsigned int *border_width*;
int *depth*;
unsigned int *class*;
Visual **visual*
unsigned long *valuemask*;
XSetWindowAttributes **attributes*;

Window XCreateSimpleWindow(*display, parent, x, y, width, height, border_width, border, background*)

Display **display*;
Window *parent*;
int *x, y*;
unsigned int *width, height*;
unsigned int *border_width*;
unsigned long *border*;
unsigned long *background*;

ARGUMENTS

attributes Specifies the structure from which the values (as specified by the value mask) are to be taken. The value mask should have the appropriate bits set to indicate which attributes have been set in the structure.

background Specifies the background pixel value of the window.

border Specifies the border pixel value of the window.

border_width Specifies the width of the created window's border in pixels.

class Specifies the created window's class. You can pass **InputOutput**, **InputOnly**, or **CopyFromParent**. A class of **CopyFromParent** means the class is taken from the parent.

depth Specifies the window's depth. A depth of **CopyFromParent** means the depth is taken from the parent.

display Specifies the connection to the X server.

parent Specifies the parent window.

valuemask Specifies which window attributes are defined in the attributes argument. This mask is the bitwise inclusive OR of the valid attribute mask bits. If valuemask is zero, the attributes are ignored and are not referenced.

visual Specifies the visual type. A visual of **CopyFromParent** means the visual type is taken from the parent.

width
height Specify the width and height, which are the created window's inside dimensions and do not include the created window's borders.

x
y Specify the x and y coordinates, which are the top-left outside corner of the window's borders and are relative to the inside of the parent window's borders.

DESCRIPTION

The **XCreateWindow** function creates an unmapped subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a **CreateNotify** event. The created window is placed on top in the stacking order with respect to siblings.

The border_width for an **InputOnly** window must be zero, or a **BadMatch** error results. For class **InputOutput**, the visual type and depth must be a combination supported for the screen, or a **BadMatch** error results. The depth need not be the same as the parent, but the parent must not be a window of class **InputOnly**, or a **BadMatch** error results. For an **InputOnly** window, the depth must be zero, and the visual must be one supported by the screen. If either condition is not met, a **BadMatch** error results. The parent window, however, may have any depth and class. If you specify any invalid window attribute for a window, a **BadMatch** error results.

The created window is not yet displayed (mapped) on the user's display. To display the window, call **XMapWindow**. The new window initially uses the same cursor as its parent. A new cursor can be defined for the new window by calling **XDefineCursor**. The window will not be visible on the screen unless it and all of its ancestors are mapped and it is not obscured by any of its ancestors.

XCreateWindow can generate **BadAlloc BadColor**, **BadCursor**, **BadMatch**, **BadPixmap**, **BadValue**, and **BadWindow** errors.

The **XCreateSimpleWindow** function creates an unmapped **InputOutput** subwindow for a specified parent window, returns the window ID of the created window, and causes the X server to generate a **CreateNotify** event. The created window is placed on top in the stacking order with respect to siblings. Any part of the window that extends outside its parent window is clipped. The border_width for an **InputOnly** window must be zero, or a **BadMatch** error results. **XCreateSimpleWindow** inherits its depth, class, and visual from its parent. All other window attributes, except background and border, have their default values.

XCreateSimpleWindow can generate **BadAlloc**, **BadMatch**, **BadValue**, and **BadWindow** errors.

STRUCTURES

The **XSetWindow**Attributes structure contains:

```
/* Window attribute value mask bits */
#define    CWBackPixmap          (1L<<0)
#define    CWBackPixel           (1L<<1)
#define    CWBorderPixmap        (1L<<2)
#define    CWBorderPixel         (1L<<3)
#define    CWBitGravity          (1L<<4)
#define    CWWinGravity          (1L<<5)
#define    CWBackingStore        (1L<<6)
#define    CWBackingPlanes       (1L<<7)
#define    CWBackingPixel        (1L<<8)
#define    CWOverrideRedirect    (1L<<9)
#define    CWSaveUnder           (1L<<10)
#define    CWEventMask           (1L<<11)
#define    CWDontPropagate       (1L<<12)
#define    CWColormap            (1L<<13)
#define    CWCursor              (1L<<14)
/* Values */
```

```
typedef struct {
        Pixmap background_pixmap;           /* background, None, or ParentRelative */
        unsigned long background_pixel;     /* background pixel */
        Pixmap border_pixmap;               /* border of the window or CopyFromParent */
        unsigned long border_pixel;         /* border pixel value */
        int bit_gravity;                    /* one of bit gravity values */
        int win_gravity;                    /* one of the window gravity values */
        int backing_store;                  /* NotUseful, WhenMapped, Always */
        unsigned long backing_planes;       /* planes to be preserved if possible */
        unsigned long backing_pixel;        /* value to use in restoring planes */
        Bool save_under;                    /* should bits under be saved? (popups) */
        long event_mask;                    /* set of events that should be saved */
        long do_not_propagate_mask;         /* set of events that should not propagate */
        Bool override_redirect;             /* boolean value for override_redirect */
        Colormap colormap;                  /* color map to be associated with window */
        Cursor cursor;                      /* cursor to be displayed (or None) */
} XSetWindowAttributes;
```

For a detailed explanation of the members of this structure, see *Xlib - C Language X Interface*.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadColor A value for a Colormap argument does not name a defined Colormap.

BadCursor A value for a Cursor argument does not name a defined Cursor.

BadMatch The values do not exist for an **InputOnly** window.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadPixmap A value for a Pixmap argument does not name a defined Pixmap.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XConfigureWindow(3), XDefineCursor(3), XDestroyWindow(3), XMapWindow(3), XRaiseWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCreateWindowEvent – CreateNotify event structure

STRUCTURES

The structure for **CreateNotify** events contains:

```
typedef struct {
        int type;                       /* CreateNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window parent;                  /* parent of the window */
        Window window;                  /* window id of window created */
        int x, y;                       /* window location */
        int width, height;              /* size of window */
        int border_width;               /* border width */
        Bool override_redirect;         /* creation should be overridden */
} XCreateWindowEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The parent member is set to the created window's parent. The window member specifies the created window. The x and y members are set to the created window's coordinates relative to the parent window's origin and indicate the position of the upper-left outside corner of the created window. The width and height members are set to the inside size of the created window (not including the border) and are always nonzero. The border_width member is set to the width of the created window's border, in pixels. The override_redirect member is set to the override-redirect attribute of the window. Window manager clients normally should ignore this window if the override_redirect member is **True**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XCrossingEvent – EnterNotify and LeaveNotify event structure

STRUCTURES

The structure for **EnterNotify** and **LeaveNotify** events contains:

```
typedef struct {
        int type;                       /* EnterNotify or LeaveNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;                  /* "event" window reported relative to */
        Window root;                    /* root window that the event occurred on */
        Window subwindow;               /* child window */
        Time time;                      /* milliseconds */
        int x, y;                       /* pointer x, y coordinates in event window */
        int x_root, y_root;             /* coordinates relative to root */
        int mode;                       /* NotifyNormal, NotifyGrab, NotifyUngrab */
        int detail;
                                        /*
                                        * NotifyAncestor, NotifyVirtual, NotifyInferior,
                                        * NotifyNonlinear,NotifyNonlinearVirtual
                                        */
        Bool same_screen;               /* same screen flag */
        Bool focus;                     /* boolean focus */
        unsigned int state;             /* key or button mask */
} XCrossingEvent;
typedef XCrossingEvent XEnterWindowEvent;
typedef XCrossingEvent XLeaveWindowEvent;
```

When you receive these events, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window on which the **EnterNotify** or **LeaveNotify** event was generated and is referred to as the event window. This is the window used by the X server to report the event, and is relative to the root window on which the event occurred. The root member is set to the root window of the screen on which the event occurred.

For a **LeaveNotify** event, if a child of the event window contains the initial position of the pointer, the subwindow component is set to that child. Otherwise, the X server sets the subwindow member to **None**. For an **EnterNotify** event, if a child of the event window contains the final pointer position, the subwindow component is set to that child or **None**.

The time member is set to the time when the event was generated and is expressed in milliseconds. The x and y members are set to the coordinates of the pointer position in the event window. This position is always the pointer's final position, not its initial position. If the event window is on the same screen as the root window, x and y are the pointer coordinates relative to the event window's origin. Otherwise, x and y are set to zero. The x_root and y_root members are set to the pointer's coordinates relative to the root window's origin at the time of the event.

The same_screen member is set to indicate whether the event window is on the same screen as the root window and can be either **True** or **False**. If **True**, the event and root windows are on the same screen. If **False**, the event and root windows are not on the same screen.

The focus member is set to indicate whether the event window is the focus window or an inferior of the focus window. The X server can set this member to either **True** or **False**. If **True**, the event window is the focus window or an inferior of the focus window. If **False**, the event window is not the focus window or an inferior of the focus window.

The state member is set to indicate the state of the pointer buttons and modifier keys just prior to the event. The X server can set this member to the bitwise inclusive OR of one or more of the button or modifier key masks: **Button1Mask**, **Button2Mask**, **Button3Mask**, **Button4Mask**, **Button5Mask**, **ShiftMask**, **LockMask**, **ControlMask**, **Mod1Mask**, **Mod2Mask**, **Mod3Mask**, **Mod4Mask**, **Mod5Mask**.

The mode member is set to indicate whether the events are normal events, pseudo-motion events when a grab activates, or pseudo-motion events when a grab deactivates. The X server can set this member to **NotifyNormal**, **NotifyGrab**, or **NotifyUngrab**.

The detail member is set to indicate the notify detail and can be **NotifyAncestor**, **NotifyVirtual**, **NotifyInferior**, **NotifyNonlinear**, or **NotifyNonlinearVirtual**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDefineCursor, XUndefineCursor - define cursors

SYNTAX

XDefineCursor(*display, w, cursor*)
Display **display*;
Window *w*;
Cursor *cursor*;

XUndefineCursor(*display, w*)
Display **display*;
Window *w*;

ARGUMENTS

cursor	Specifies the cursor that is to be displayed or **None**.
display	Specifies the connection to the X server.
w	Specifies the window.

DESCRIPTION

If a cursor is set, it will be used when the pointer is in the window. If the cursor is **None**, it is equivalent to **XUndefineCursor**.

XDefineCursor can generate **BadCursor** and **BadWindow** errors.

The **XUndefineCursor** undoes the effect of a previous **XDefineCursor** for this window. When the pointer is in the window, the parent's cursor will now be used. On the root window, the default cursor is restored.

XUndefineCursor can generate a **BadWindow** error.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadCursor	A value for a Cursor argument does not name a defined Cursor.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XCreateFontCursor(3), XRecolorCursor(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDestroyWindow, XDestroySubwindows – destroy windows

SYNTAX

XDestroyWindow(*display, w*)
Display **display*;
Window *w*;

XDestroySubwindows(*display, w*)
Display **display*;
Window *w*;

ARGUMENTS

display	Specifies the connection to the X server.
w	Specifies the window.

DESCRIPTION

The **XDestroyWindow** function destroys the specified window as well as all of its subwindows and causes the X server to generate a **DestroyNotify** event for each window. The window should never be referenced again. If the window specified by the w argument is mapped, it is unmapped automatically. The ordering of the **DestroyNotify** events is such that for any given window being destroyed, **DestroyNotify** is generated on any inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is not otherwise constrained. If the window you specified is a root window, no windows are destroyed. Destroying a mapped window will generate **Expose** events on other windows that were obscured by the window being destroyed.

XDestroyWindow can generate a **BadWindow** error.

The **XDestroySubwindows** function destroys all inferior windows of the specified window, in bottom-to-top stacking order. It causes the X server to generate a **DestroyNotify** event for each window. If any mapped subwindows were actually destroyed, **XDestroySubwindows** causes the X server to generate **Expose** events on the specified window. This is much more efficient than deleting many windows one at a time because much of the work need be performed only once for all of the windows, rather than for each window. The subwindows should never be referenced again.

XDestroySubwindows can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XConfigureWindow(3), XCreateWindow(3), XMapWindow(3), XRaiseWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDestroyWindowEvent – DestroyNotify event structure

STRUCTURES

The structure for **DestroyNotify** events contains:

```
typedef struct {
        int type;                       /* DestroyNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
} XDestroyWindowEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the destroyed window or to its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. The window member is set to the window that is destroyed.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawArc, XDrawArcs, XArc – draw arcs and arc structure

SYNTAX

XDrawArc(*display, d, gc, x, y, width, height, angle1, angle2*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
unsigned int *width, height*;
int *angle1, angle2*;

XDrawArcs(*display, d, gc, arcs, narcs*)
Display **display*;
Drawable *d*;
GC *gc*;
XArc **arcs*;
int *narcs*;

ARGUMENTS

angle1	Specifies the start of the arc relative to the three-o'clock position from the center, in units of degrees * 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.
arcs	Specifies a pointer to an array of arcs.
d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
narcs	Specifies the number of arcs in the array.
width *height*	Specify the width and height, which are the major and minor axes of the arc.
x *y*	Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the bounding rectangle.

DESCRIPTION

XDrawArc draws a single circular or elliptical arc, and **XDrawArcs** draws multiple circular or elliptical arcs. Each arc is specified by a rectangle and two angles. The center of the circle or ellipse is the center of the rectangle, and the major and minor axes are specified by the width and height. Positive angles indicate counterclockwise motion, and negative angles indicate clockwise motion. If the magnitude of angle2 is greater than 360 degrees, **XDrawArc** or **XDrawArcs** truncates it to 360 degrees.

For an arc specified as [$x, y, width, height, angle1, angle2$], the origin of the major and minor axes is at $[x + \frac{width}{2}, y + \frac{height}{2}]$, and the infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at $[x, y + \frac{height}{2}]$ and $[x + width, y + \frac{height}{2}]$ and intersects the vertical axis at $[x + \frac{width}{2}, y]$ and $[x + \frac{width}{2}, y + height]$. These coordinates can be fractional and so are not truncated to discrete coordinates. The path should be defined by the ideal mathematical path. For a wide line with line-width lw, the bounding outlines for filling are given by the two infinitely thin paths consisting of all points whose perpendicular distance from the

path of the circle/ellipse is equal to lw/2 (which may be a fractional value). The cap-style and join-style are applied the same as for a line corresponding to the tangent of the circle/ellipse at the endpoint.

For an arc specified as [*x*, *y*, *width*, *height*, *angle*1, *angle*2], the angles must be specified in the effectively skewed coordinate system of the ellipse (for a circle, the angles and coordinate systems are identical). The relationship between these angles and angles expressed in the normal coordinate system of the screen (as measured with a protractor) is as follows:

$$\text{skewed-angle} = \text{atan}\left(\tan(\text{normal-angle}) * \frac{width}{height}\right) + adjust$$

The skewed-angle and normal-angle are expressed in radians (rather than in degrees scaled by 64) in the range $[0, 2\pi]$ and where atan returns a value in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$ and adjust is:

0	for normal-angle in the range $[0, \frac{\pi}{2}]$
π	for normal-angle in the range $[\frac{\pi}{2}, \frac{3\pi}{2}]$
2π	for normal-angle in the range $[\frac{3\pi}{2}, 2\pi]$

For any given arc, **XDrawArc** and **XDrawArcs** do not draw a pixel more than once. If two arcs join correctly and if the line-width is greater than zero and the arcs intersect, **XDrawArc** and **XDrawArcs** do not draw a pixel more than once. Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times. Specifying an arc with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

If the last point in one arc coincides with the first point in the following arc, the two arcs will join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs will join correctly. By specifying one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio.

Both functions use these GC components: function, plane-mask, line-width, line-style, cap-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawArc and **XDrawArcs** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

STRUCTURES

The **XArc** structure contains:

```
typedef struct {
        short x, y;
        unsigned short width, height;
        short angle1, angle2;           /* Degrees * 64 */
} XArc;
```

All x and y members are signed integers. The width and height members are 16-bit unsigned integers. You should be careful not to generate coordinates and sizes out of the 16-bit ranges, because the protocol only has 16-bit fields for these values.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

SEE ALSO

XDrawLine(3), XDrawPoint(3), XDrawRectangle(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawImageString, XDrawImageString16 – draw image text

SYNTAX

XDrawImageString(*display, d, gc, x, y, string, length*)

```
Display *display;
Drawable d;
GC gc;
int x, y;
char *string;
int length;
```

XDrawImageString16(*display, d, gc, x, y, string, length*)

```
Display *display;
Drawable d;
GC gc;
int x, y;
XChar2b *string;
int length;
```

ARGUMENTS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
length	Specifies the number of characters in the string argument.
string	Specifies the character string.
x *y*	Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.

DESCRIPTION

The **XDrawImageString16** function is similar to **XDrawImageString** except that it uses 2-byte or 16-bit characters. Both functions also use both the foreground and background pixels of the GC in the destination.

The effect is first to fill a destination rectangle with the background pixel defined in the GC and then to paint the text with the foreground pixel. The upper-left corner of the filled rectangle is at:

[x, y – font-ascent]

The width is:

overall-width

The height is:

font-ascent + font-descent

The overall-width, font-ascent, and font-descent are as would be returned by **XQueryTextExtents** using gc and string. The function and fill-style defined in the GC are ignored for these functions. The effective function is **GXcopy**, and the effective fill-style is **FillSolid**.

For fonts defined with 2-byte matrix indexing and used with **XDrawImageString**, each byte is used as a byte2 with a byte1 of zero.

Both functions use these GC components: plane-mask, foreground, background, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

XDrawImageString and **XDrawImageString16** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

SEE ALSO

XDrawString(3), XDrawText(3), XLoadFont(3), XTextExtents(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawLine, XDrawLines, XDrawSegments, XSegment – draw lines, polygons, and line structure

SYNTAX

XDrawLine(*display, d, gc, x1, y1, x2, y2*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x1, y1, x2, y2*;

XDrawLines(*display, d, gc, points, npoints, mode*)
Display **display*;
Drawable *d*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *mode*;

XDrawSegments(*display, d, gc, segments, nsegments*)
Display **display*;
Drawable *d*;
GC *gc*;
XSegment **segments*;
int *nsegments*;

ARGUMENTS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
mode	Specifies the coordinate mode. You can pass **CoordModeOrigin** or **CoordModePrevious**.
npoints	Specifies the number of points in the array.
nsegments	Specifies the number of segments in the array.
points	Specifies a pointer to an array of points.
segments	Specifies a pointer to an array of segments.
x1 *y1* *x2* *y2*	Specify the points (x1, y1) and (x2, y2) to be connected.

DESCRIPTION

The **XDrawLine** function uses the components of the specified GC to draw a line between the specified set of points (x1, y1) and (x2, y2). It does not perform joining at coincident endpoints. For any given line, **XDrawLine** does not draw a pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times.

The **XDrawLines** function uses the components of the specified GC to draw npoints–1 lines between each pair of points (point[i], point[i+1]) in the array of **XPoint** structures. It draws the lines in the order listed in the array. The lines join correctly at all intermediate points, and if the first and last points coincide, the first and last lines also join correctly. For any given line, **XDrawLines** does not draw a pixel more than once. If thin (zero line-width) lines intersect, the intersecting pixels are drawn multiple times. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire **PolyLine** protocol request were a single, filled shape. **CoordModeOrigin** treats all coordinates as relative to the origin, and **CoordModePrevious**

treats all coordinates after the first as relative to the previous point.

The **XDrawSegments** function draws multiple, unconnected lines. For each segment, **XDrawSegments** draws a line between (x1, y1) and (x2, y2). It draws the lines in the order listed in the array of **XSegment** structures and does not perform joining at coincident endpoints. For any given line, **XDrawSegments** does not draw a pixel more than once. If lines intersect, the intersecting pixels are drawn multiple times.

All three functions use these GC components: function, plane-mask, line-width, line-style, cap-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. The **XDrawLines** function also uses the join-style GC component. All three functions also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawLine, **XDrawLines**, and **XDrawSegments** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors. **XDrawLines** can also generate a **BadValue** error.

STRUCTURES

The **XSegment** structure contains:

```
typedef struct {
        short x1, y1, x2, y2;
} XSegment;
```

All x and y members are signed integers. The width and height members are 16-bit unsigned integers. You should be careful not to generate coordinates and sizes out of the 16-bit ranges, because the protocol only has 16-bit fields for these values.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XDrawArc(3), XDrawPoint(3), XDrawRectangle(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawPoint, XDrawPoints, XPoint – draw points and points structure

SYNTAX

XDrawPoint(*display, d, gc, x, y*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;

XDrawPoints(*display, d, gc, points, npoints, mode*)
Display **display*;
Drawable *d*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *mode*;

ARGUMENTS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
mode	Specifies the coordinate mode. You can pass **CoordModeOrigin** or **CoordModePrevious**.
npoints	Specifies the number of points in the array.
points	Specifies a pointer to an array of points.
x *y*	Specify the x and y coordinates where you want the point drawn.

DESCRIPTION

The **XDrawPoint** function uses the foreground pixel and function components of the GC to draw a single point into the specified drawable; **XDrawPoints** draws multiple points this way. **CoordModeOrigin** treats all coordinates as relative to the origin, and **CoordModePrevious** treats all coordinates after the first as relative to the previous point. **XDrawPoints** draws the points in the order listed in the array.

Both functions use these GC components: function, plane-mask, foreground, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask.

XDrawPoint can generate **BadDrawable**, **BadGC**, and **BadMatch** errors. **XDrawPoints** can generate **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue** errors.

STRUCTURES

The **XPoint** structure contains:

```
typedef struct {
        short x, y;
} XPoint;
```

All x and y members are signed integers. The width and height members are 16-bit unsigned integers. You should be careful not to generate coordinates and sizes out of the 16-bit ranges, because the protocol only has 16-bit fields for these values.

DIAGNOSTICS

BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadGC	A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XDrawArc(3), XDrawLine(3), XDrawRectangle(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawRectangle, XDrawRectangles, XRectangle – draw rectangles and rectangles structure

SYNTAX

XDrawRectangle(*display, d, gc, x, y, width, height*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
unsigned int *width, height*;

XDrawRectangles(*display, d, gc, rectangles, nrectangles*)
Display **display*;
Drawable *d*;
GC *gc*;
XRectangle *rectangles*[];
int *nrectangles*;

ARGUMENTS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
nrectangles	Specifies the number of rectangles in the array.
rectangles	Specifies a pointer to an array of rectangles.
width *height*	Specify the width and height, which specify the dimensions of the rectangle.
x *y*	Specify the x and y coordinates, which specify the upper-left corner of the rectangle.

DESCRIPTION

The **XDrawRectangle** and **XDrawRectangles** functions draw the outlines of the specified rectangle or rectangles as if a five-point **PolyLine** protocol request were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height] [x,y]

For the specified rectangle or rectangles, these functions do not draw a pixel more than once. **XDrawRectangles** draws the rectangles in the order listed in the array. If rectangles intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, line-width, line-style, join-style, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, tile-stipple-y-origin, dash-offset, and dash-list.

XDrawRectangle and **XDrawRectangles** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

STRUCTURES

The **XRectangle** structure contains:

```
typedef struct {
        short x, y;
        unsigned short width, height;
} XRectangle;
```

All x and y members are signed integers. The width and height members are 16-bit unsigned integers. You should be careful not to generate coordinates and sizes out of the 16-bit ranges, because the protocol only has 16-bit fields for these values.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

SEE ALSO

XDrawArc(3), XDrawLine(3), XDrawPoint(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawString, XDrawString16 – draw text characters

SYNTAX

XDrawString(*display, d, gc, x, y, string, length*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
char **string*;
int *length*;

XDrawString16(*display, d, gc, x, y, string, length*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
XChar2b **string*;
int *length*;

ARGUMENTS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
length	Specifies the number of characters in the string argument.
string	Specifies the character string.
x *y*	Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.

DESCRIPTION

Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. For fonts defined with 2-byte matrix indexing and used with **XDrawString16**, each byte is used as a byte2 with a byte1 of zero.

Both functions use these GC components: function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XDrawString and **XDrawString16** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

DIAGNOSTICS

BadDrawable	A value for a Drawable argument does not name a defined Window or Pixmap.
BadGC	A value for a GContext argument does not name a defined GContext.
BadMatch	An **InputOnly** window is used as a Drawable.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

SEE ALSO

XDrawImageString(3), XDrawText(3), XLoadFont(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XDrawText, XDrawText16, XTextItem, XTextItem16 – draw polytext text and text drawing structures

SYNTAX

XDrawText(*display, d, gc, x, y, items, nitems*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
XTextItem **items*;
int *nitems*;

XDrawText16(*display, d, gc, x, y, items, nitems*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
XTextItem16 **items*;
int *nitems*;

ARGUMENTS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
items	Specifies a pointer to an array of text items.
nitems	Specifies the number of text items in the array.
x *y*	Specify the x and y coordinates, which are relative to the origin of the specified drawable and define the origin of the first character.

DESCRIPTION

The **XDrawText16** function is similar to **XDrawText** except that it uses 2-byte or 16-bit characters. Both functions allow complex spacing and font shifts between counted strings.

Each text item is processed in turn. A font member other than **None** in an item causes the font to be stored in the GC and used for subsequent text. A text element delta specifies an additional change in the position along the x axis before the string is drawn. The delta is always added to the character origin and is not dependent on any characteristics of the font. Each character image, as defined by the font in the GC, is treated as an additional mask for a fill operation on the drawable. The drawable is modified only where the font character has a bit set to 1. If a text item generates a **BadFont** error, the previous text items may have been drawn.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each **XChar2b** structure is interpreted as a 16-bit number with byte1 as the most-significant byte.

Both functions use these GC components: function, plane-mask, fill-style, font, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XDrawText and **XDrawText16** can generate **BadDrawable**, **BadFont**, **BadGC**, and **BadMatch** errors.

STRUCTURES

The **XTextItem** and **XTextItem16** structures contain:

```
typedef struct {
        char *chars;                    /* pointer to string */
        int nchars;                     /* number of characters */
        int delta;                      /* delta between strings */
        Font font;                      /* Font to print it in, None don't change */
} XTextItem;

typedef struct {
        XChar2b *chars;                 /* pointer to two-byte characters */
        int nchars;                     /* number of characters */
        int delta;                      /* delta between strings */
        Font font;                      /* font to print it in, None don't change */
} XTextItem16;
```

If the font member is not **None**, the font is changed before printing and also is stored in the GC. If an error was generated during text drawing, the previous items may have been drawn. The baseline of the characters are drawn starting at the x and y coordinates that you pass in the text drawing functions.

For example, consider the background rectangle drawn by **XDrawImageString**. If you want the upper-left corner of the background rectangle to be at pixel coordinate (x,y), pass the (x,y + ascent) as the baseline origin coordinates to the text functions. The ascent is the font ascent, as given in the **XFontStruct** structure. If you want the lower-left corner of the background rectangle to be at pixel coordinate (x,y), pass the (x,y - descent + 1) as the baseline origin coordinates to the text functions. The descent is the font descent, as given in the **XFontStruct** structure.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadFont A value for a Font or GContext argument does not name a defined Font.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

SEE ALSO

XDrawImageString(3), XDrawString(3), XLoadFont(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XEmptyRegion, XEqualRegion, XPointInRegion, XRectInRegion – determine if regions are empty or equal

SYNTAX

Bool XEmptyRegion(*r*)
Region *r*;

Bool XEqualRegion(*r1*, *r2*)
Region *r1*, *r2*;

Bool XPointInRegion(*r*, *x*, *y*)
Region *r*;
int *x*, *y*;

int XRectInRegion(*r*, *x*, *y*, *width*, *height*)
Region *r*;
int *x*, *y*;
unsigned int *width*, *height*;

ARGUMENTS

r	Specifies the region.
r1 *r2*	Specify the two regions.
width *height*	Specify the width and height, which define the rectangle.
x *y*	Specify the x and y coordinates, which define the point or the coordinates of the upper-left corner of the rectangle.

DESCRIPTION

The **XEmptyRegion** function returns **True** if the region is empty.

The **XEqualRegion** function returns **True** if the two regions have the same offset, size, and shape.

The **XPointInRegion** function returns **True** if the point (x, y) is contained in the region r.

The **XRectInRegion** function returns **RectangleIn** if the rectangle is entirely in the specified region, **RectangleOut** if the rectangle is entirely out of the specified region, and **RectanglePart** if the rectangle is partially in the specified region.

SEE ALSO

XCreateRegion(3), XIntersectRegion(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XErrorEvent – X error event structure

STRUCTURES

The **XErrorEvent** structure contains:

```
typedef struct {
        int type;
        Display *display;               /* Display the event was read from */
        unsigned long serial;           /* serial number of failed request */
        unsigned char error_code;       /* error code of failed request */
        unsigned char request_code;     /* Major op-code of failed request */
        unsigned char minor_code;       /* Minor op-code of failed request */
        XID resourceid;                 /* resource id */
} XErrorEvent;
```

When you receive this event, the structure members are set as follows.

The serial member is the number of requests, starting from one, sent over the network connection since it was opened. It is the number that was the value of **NextRequest** immediately before the failing call was made. The request_code member is a protocol request of the procedure that failed, as defined in <**X11/Xproto.h**>.

SEE ALSO

AllPlanes(3), XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XExposeEvent – Expose event structure

STRUCTURES

The structure for **Expose** events contains:

```
typedef struct {
        int type;                       /* Expose */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        int x, y;
        int width, height;
        int count;                      /* if nonzero, at least this many more */
} XExposeEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the exposed (damaged) window. The x and y members are set to the coordinates relative to the window's origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of **Expose** events that are to follow. If count is zero, no more **Expose** events follow for this window. However, if count is nonzero, at least that number of **Expose** events (and possibly more) follow for this window. Simple applications that do not want to optimize redisplay by distinguishing between subareas of its window can just ignore all **Expose** events with nonzero counts and perform full redisplays on events with zero counts.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XFillRectangle, XFillRectangles, XFillPolygon, XFillArc, XFillArcs - fill rectangles, polygons, or arcs

SYNTAX

XFillRectangle(*display, d, gc, x, y, width, height*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
unsigned int *width, height*;

XFillRectangles(*display, d, gc, rectangles, nrectangles*)
Display **display*;
Drawable *d*;
GC *gc*;
XRectangle **rectangles*;
int *nrectangles*;

XFillPolygon(*display, d, gc, points, npoints, shape, mode*)
Display **display*;
Drawable *d*;
GC *gc*;
XPoint **points*;
int *npoints*;
int *shape*;
int *mode*;

XFillArc(*display, d, gc, x, y, width, height, angle1, angle2*)
Display **display*;
Drawable *d*;
GC *gc*;
int *x, y*;
unsigned int *width, height*;
int *angle1, angle2*;

XFillArcs(*display, d, gc, arcs, narcs*)
Display **display*;
Drawable *d*;
GC *gc*;
XArc **arcs*;
int *narcs*;

ARGUMENTS

angle1	Specifies the start of the arc relative to the three-o'clock position from the center, in units of degrees * 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc, in units of degrees * 64.
arcs	Specifies a pointer to an array of arcs.
d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the GC.
mode	Specifies the coordinate mode. You can pass **CoordModeOrigin** or **CoordModePrevious**.

narcs	Specifies the number of arcs in the array.
npoints	Specifies the number of points in the array.
nrectangles	Specifies the number of rectangles in the array.
points	Specifies a pointer to an array of points.
rectangles	Specifies a pointer to an array of rectangles.
shape	Specifies a shape that helps the server to improve performance. You can pass **Complex**, **Convex**, or **Nonconvex**.
width *height*	Specify the width and height, which are the dimensions of the rectangle to be filled or the major and minor axes of the arc.
x *y*	Specify the x and y coordinates, which are relative to the origin of the drawable and specify the upper-left corner of the rectangle.

DESCRIPTION

The **XFillRectangle** and **XFillRectangles** functions fill the specified rectangle or rectangles as if a four-point **FillPolygon** protocol request were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height]

Each function uses the x and y coordinates, width and height dimensions, and GC you specify.

XFillRectangles fills the rectangles in the order listed in the array. For any given rectangle, **XFillRectangle** and **XFillRectangles** do not draw a pixel more than once. If rectangles intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, fill-style, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillRectangle and **XFillRectangles** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

XFillPolygon fills the region closed by the specified path. The path is closed automatically if the last point in the list does not coincide with the first point. **XFillPolygon** does not draw a pixel of the region more than once. **CoordModeOrigin** treats all coordinates as relative to the origin, and **CoordModePrevious** treats all coordinates after the first as relative to the previous point.

Depending on the specified shape, the following occurs:

- If shape is **Complex**, the path may self-intersect. Note that contiguous coincident points in the path are not treated as self-intersection.
- If shape is **Convex**, for every pair of points inside the polygon, the line segment connecting them does not intersect the path. If known by the client, specifying **Convex** can improve performance. If you specify **Convex** for a path that is not convex, the graphics results are undefined.
- If shape is **Nonconvex**, the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifying **Nonconvex** instead of **Complex** may improve performance. If you specify **Nonconvex** for a self-intersecting path, the graphics results are undefined.

The fill-rule of the GC controls the filling behavior of self-intersecting polygons.

This function uses these GC components: function, plane-mask, fill-style, fill-rule, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. It also uses these GC mode-dependent

components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillPolygon can generate **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue** errors.

For each arc, **XFillArc** or **XFillArcs** fills the region closed by the infinitely thin path described by the specified arc and, depending on the arc-mode specified in the GC, one or two line segments. For **ArcChord**, the single line segment joining the endpoints of the arc is used. For **ArcPieSlice**, the two line segments joining the endpoints of the arc with the center point are used. **XFillArcs** fills the arcs in the order listed in the array. For any given arc, **XFillArc** and **XFillArcs** do not draw a pixel more than once. If regions intersect, the intersecting pixels are drawn multiple times.

Both functions use these GC components: function, plane-mask, fill-style, arc-mode, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. They also use these GC mode-dependent components: foreground, background, tile, stipple, tile-stipple-x-origin, and tile-stipple-y-origin.

XFillArc and **XFillArcs** can generate **BadDrawable**, **BadGC**, and **BadMatch** errors.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XDrawArc(3), XDrawPoint(3), XDrawRectangle(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XFlush, XSync, XEventsQueued, XPending – handle output buffer or event queue

SYNTAX

XFlush(*display*)
Display **display*;

XSync(*display, discard*)
Display **display*;
Bool *discard*;

int XEventsQueued(*display, mode*)
Display **display*;
int *mode*;

int XPending(*display*)
Display **display*;

ARGUMENTS

discard	Specifies a Boolean value that indicates whether **XSync** discards all events on the event queue.
display	Specifies the connection to the X server.
mode	Specifies the mode. You can pass **QueuedAlready**, **QueuedAfterFlush**, or **QueuedAfterReading**.

DESCRIPTION

The **XFlush** function flushes the output buffer. Most client applications need not use this function because the output buffer is automatically flushed as needed by calls to **XPending**, **XNextEvent**, and **XWindowEvent**. Events generated by the server may be enqueued into the library's event queue.

The **XSync** function flushes the output buffer and then waits until all requests have been received and processed by the X server. Any errors generated must be handled by the error handler. For each error event received by Xlib, **XSync** calls the client application's error handling routine (see section 8.12.2). Any events generated by the server are enqueued into the library's event queue.

Finally, if you passed **False**, **XSync** does not discard the events in the queue. If you passed **True**, **XSync** discards all events in the queue, including those events that were on the queue before **XSync** was called. Client applications seldom need to call **XSync**.

If mode is **QueuedAlready**, **XEventsQueued** returns the number of events already in the event queue (and never performs a system call). If mode is **QueuedAfterFlush**, **XEventsQueued** returns the number of events already in the queue if the number is nonzero. If there are no events in the queue, **XEventsQueued** flushes the output buffer, attempts to read more events out of the application's connection, and returns the number read. If mode is **QueuedAfterReading**, **XEventsQueued** returns the number of events already in the queue if the number is nonzero. If there are no events in the queue, **XEventsQueued** attempts to read more events out of the application's connection without flushing the output buffer and returns the number read.

XEventsQueued always returns immediately without I/O if there are events already in the queue. **XEventsQueued** with mode **QueuedAfterFlush** is identical in behavior to **XPending**. **XEventsQueued** with mode **QueuedAlready** is identical to the **XQLength** function.

The **XPending** function returns the number of events that have been received from the X server but have not been removed from the event queue. **XPending** is identical to **XEventsQueued** with the mode **QueuedAfterFlush** specified.

SEE ALSO

AllPlanes(3), XIfEvent(3), XNextEvent(3), XPutBackEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XFocusChangeEvent – FocusIn and FocusOut event structure

STRUCTURES

The structure for **FocusIn** and **FocusOut** events contains:

```
typedef struct {
        int type;                       /* FocusIn or FocusOut */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;                  /* window of event */
        int mode;                       /* NotifyNormal, NotifyGrab, NotifyUngrab */
        int detail;
                                        /*
                                        * NotifyAncestor, NotifyVirtual, NotifyInferior,
                                        * NotifyNonlinear,NotifyNonlinearVirtual, NotifyPointer,
                                        * NotifyPointerRoot, NotifyDetailNone
                                        */
} XFocusChangeEvent;
typedef XFocusChangeEvent XFocusInEvent;
typedef XFocusChangeEvent XFocusOutEvent;
```

When you receive these events, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window on which the **FocusIn** or **FocusOut** event was generated. This is the window used by the X server to report the event. The mode member is set to indicate whether the focus events are normal focus events, focus events while grabbed, focus events when a grab activates, or focus events when a grab deactivates. The X server can set the mode member to **NotifyNormal**, **NotifyWhileGrabbed**, **NotifyGrab**, or **NotifyUngrab**.

All **FocusOut** events caused by a window unmap are generated after any **UnmapNotify** event; however, the X protocol does not constrain the ordering of **FocusOut** events with respect to generated **EnterNotify**, **LeaveNotify**, **VisibilityNotify**, and **Expose** events.

Depending on the event mode, the detail member is set to indicate the notify detail and can be **NotifyAncestor**, **NotifyVirtual**, **NotifyInferior**, **NotifyNonlinear**, **NotifyNonlinearVirtual**, **NotifyPointer**, **NotifyPointerRoot**, or **NotifyDetailNone**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XFree, XNoOp – free client data

SYNTAX

XFree(*data*)
char **data*;

XNoOp(*display*)
Display **display*;

ARGUMENTS

display	Specifies the connection to the X server.
data	Specifies a pointer to the data that is to be freed.

DESCRIPTION

The **XFree** function is a general-purpose Xlib routine that frees the specified data. You must use it to free any objects that were allocated by Xlib.

The **XNoOp** function sends a **NoOperation** protocol request to the X server, thereby exercising the connection.

SEE ALSO

Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGetDefault, XResourceManagerString – get X program defaults

SYNTAX

char *XGetDefault(*display*, *program*, *option*)
Display **display*;
char **program*;
char **option*;

char *XResourceManagerString(*display*)
Display **display*;

ARGUMENTS

display	Specifies the connection to the X server.
option	Specifies the option name.
program	Specifies the program name for the Xlib defaults (usually argv[0] of the main program).

DESCRIPTION

The **XGetDefault** function returns the value NULL if the option name specified in this argument does not exist for the program. The strings returned by **XGetDefault** are owned by Xlib and should not be modified or freed by the client.

The **XResourceManagerString** returns the RESOURCE_MANAGER property from the server's root window of screen zero, which was returned when the connection was opened using **XOpenDisplay**. Note that the property value must be in a format that is acceptable to **XrmGetStringDatabase**.

SEE ALSO

XOpenDisplay(3), XrmGetSearchList(3), XrmMergeDatabases(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGetVisualInfo, XMatchVisualInfo, XVisualIDFromVisual, XVisualInfo – obtain visual information and visual structure

SYNTAX

XVisualInfo *XGetVisualInfo(*display, vinfo_mask, vinfo_template, nitems_return*)
Display **display*;
long *vinfo_mask*;
XVisualInfo **vinfo_template*;
int **nitems_return*;

Status XMatchVisualInfo(*display, screen, depth, class, vinfo_return*)
Display **display*;
int *screen*;
int *depth*;
int *class*;
XVisualInfo **vinfo_return*;

VisualID XVisualIDFromVisual(*visual*)
Visual **visual*;

ARGUMENTS

class	Specifies the class of the screen.
depth	Specifies the depth of the screen.
display	Specifies the connection to the X server.
nitems_return	Returns the number of matching visual structures.
screen	Specifies the screen.
visual	Specifies the visual type.
vinfo_mask	Specifies the visual mask value.
vinfo_return	Returns the matched visual information.
vinfo_template	Specifies the visual attributes that are to be used in matching the visual structures.

DESCRIPTION

The **XGetVisualInfo** function returns a list of visual structures that match the attributes specified by vinfo_template. If no visual structures match the template using the specified vinfo_mask, **XGetVisualInfo** returns a NULL. To free the data returned by this function, use **XFree**.

The **XMatchVisualInfo** function returns the visual information for a visual that matches the specified depth and class for a screen. Because multiple visuals that match the specified depth and class can exist, the exact visual chosen is undefined. If a visual is found, **XMatchVisualInfo** returns nonzero and the information on the visual to vinfo_return. Otherwise, when a visual is not found, **XMatchVisualInfo** returns zero.

The **XVisualIDFromVisual** function returns the visual ID for the specified visual type.

STRUCTURES

The **XVisualInfo** structure contains:

```
/* Visual information mask bits */
#define    VisualNoMask            0x0
#define    VisualIDMask            0x1
#define    VisualScreenMask        0x2
#define    VisualDepthMask         0x4
```

```
#define     VisualClassMask              0x8
#define     VisualRedMaskMask            0x10
#define     VisualGreenMaskMask          0x20
#define     VisualBlueMaskMask           0x40
#define     VisualColormapSizeMask       0x80
#define     VisualBitsPerRGBMask         0x100
#define     VisualAllMask                0x1FF
/* Values */

typedef struct {
        Visual *visual;
        VisualID visualid;
        int screen;
        unsigned int depth;
        int class;
        unsigned long red_mask;
        unsigned long green_mask;
        unsigned long blue_mask;
        int colormap_size;
        int bits_per_rgb;
} XVisualInfo;
```

SEE ALSO

XFree(3)

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGetWindowAttributes, XGetGeometry, XWindowAttributes - get current window attribute or geometry and current window attributes structure

SYNTAX

Status XGetWindowAttributes(*display, w, window_attributes_return*)
Display **display*;
Window *w*;
XWindowAttributes **window_attributes_return*;

Status XGetGeometry(*display, d, root_return, x_return, y_return, width_return, height_return, border_width_return, depth_return*)
Display **display*;
Drawable *d*;
Window **root_return*;
int **x_return*, **y_return*;
unsigned int **width_return*, **height_return*;
unsigned int **border_width_return*;
unsigned int **depth_return*;

ARGUMENTS

border_width_return
Returns the border width in pixels.

d Specifies the drawable, which can be a window or a pixmap.

depth_return Returns the depth of the drawable (bits per pixel for the object).

display Specifies the connection to the X server.

root_return Returns the root window.

w Specifies the window whose current attributes you want to obtain.

width_return
height_return Return the drawable's dimensions (width and height).

window_attributes_return
Returns the specified window's attributes in the **XWindowAttributes** structure.

x_return
y_return Return the x and y coordinates that define the location of the drawable. For a window, these coordinates specify the upper-left outer corner relative to its parent's origin. For pixmaps, these coordinates are always zero.

DESCRIPTION

The **XGetWindowAttributes** function returns the current attributes for the specified window to an **XWindowAttributes** structure.

XGetWindowAttributes can generate **BadDrawable** and **BadWindow** errors.

The **XGetGeometry** function returns the root window and the current geometry of the drawable. The geometry of the drawable includes the x and y coordinates, width and height, border width, and depth. These are described in the argument list. It is legal to pass to this function a window whose class is **InputOnly**.

STRUCTURES

The **XWindowAttributes** structure contains:

```
typedef struct {
        int x, y;                       /* location of window */
        int width, height;              /* width and height of window */
        int border_width;               /* border width of window */
```

```
    int depth;                          /* depth of window */
    Visual *visual;                     /* the associated visual structure */
    Window root;                        /* root of screen containing window */
    int class;                          /* InputOutput, InputOnly*/
    int bit_gravity;                    /* one of the bit gravity values */
    int win_gravity;                    /* one of the window gravity values */
    int backing_store;                  /* NotUseful, WhenMapped, Always */
    unsigned long backing_planes;       /* planes to be preserved if possible */
    unsigned long backing_pixel;        /* value to be used when restoring planes */
    Bool save_under;                    /* boolean, should bits under be saved? */
    Colormap colormap;                  /* color map to be associated with window */
    Bool map_installed;                 /* boolean, is color map currently installed*/
    int map_state;                      /* IsUnmapped, IsUnviewable, IsViewable */
    long all_event_masks;               /* set of events all people have interest in*/
    long your_event_mask;               /* my event mask */
    long do_not_propagate_mask;         /* set of events that should not propagate */
    Bool override_redirect;             /* boolean value for override-redirect */
    Screen *screen;                     /* back pointer to correct screen */
} XWindowAttributes;
```

The x and y members are set to the upper-left outer corner relative to the parent window's origin. The width and height members are set to the inside size of the window, not including the border. The border_width member is set to the window's border width in pixels. The depth member is set to the depth of the window (that is, bits per pixel for the object). The visual member is a pointer to the screen's associated **Visual** structure. The root member is set to the root window of the screen containing the window. The class member is set to the window's class and can be either **InputOutput** or **InputOnly**.

The bit_gravity member is set to the window's bit gravity and can be one of the following:

ForgetGravity	**EastGravity**
NorthWestGravity	**SouthWestGravity**
NorthGravity	**SouthGravity**
NorthEastGravity	**SouthEastGravity**
WestGravity	**StaticGravity**
CenterGravity	

The win_gravity member is set to the window's window gravity and can be one of the following:

UnmapGravity	**EastGravity**
NorthWestGravity	**SouthWestGravity**
NorthGravity	**SouthGravity**
NorthEastGravity	**SouthEastGravity**
WestGravity	**StaticGravity**
CenterGravity	

For additional information on gravity, see section 3.3.

The backing_store member is set to indicate how the X server should maintain the contents of a window and can be **WhenMapped**, **Always**, or **NotUseful**. The backing_planes member is set to indicate (with bits set to 1) which bit planes of the window hold dynamic data that must be preserved in backing_stores and during save_unders. The backing_pixel member is set to indicate what values to use for planes not set in backing_planes.

The save_under member is set to **True** or **False**. The colormap member is set to the colormap for the specified window and can be a colormap ID or **None**. The map_installed member is set to indicate whether the colormap is currently installed and can be **True** or **False**. The map_state member is set to indicate the state of the window and can be **IsUnmapped**, **IsUnviewable**, or

IsViewable. **IsUnviewable** is used if the window is mapped but some ancestor is unmapped.

The all_event_masks member is set to the bitwise inclusive OR of all event masks selected on the window by all clients. The your_event_mask member is set to the bitwise inclusive OR of all event masks selected by the querying client. The do_not_propagate_mask member is set to the bitwise inclusive OR of the set of events that should not propagate.

The override_redirect member is set to indicate whether this window overrides structure control facilities and can be **True** or **False**. Window manager clients should ignore the window if this member is **True**.

The screen member is set to a screen pointer that gives you a back pointer to the correct screen. This makes it easier to obtain the screen information without having to loop over the root window fields to see which field matches.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XQueryPointer(3), XQueryTree(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGetWindowProperty, XListProperties, XChangeProperty, XRotateWindowProperties, XDeleteProperty – obtain and change window properties

SYNTAX

int XGetWindowProperty(*display, w, property, long_offset, long_length, delete, req_type, actual_type_return, actual_format_return, nitems_return, bytes_after_return, prop_return*)
Display **display*;
Window *w*;
Atom *property*;
long *long_offset*, *long_length*;
Bool *delete*;
Atom *req_type*;
Atom **actual_type_return*;
int **actual_format_return*;
unsigned long **nitems_return*;
unsigned long **bytes_after_return*;
unsigned char ***prop_return*;

Atom *XListProperties(*display, w, num_prop_return*)
Display **display*;
Window *w*;
int **num_prop_return*;

XChangeProperty(*display, w, property, type, format, mode, data, nelements*)
Display **display*;
Window *w*;
Atom *property, type*;
int *format*;
int *mode*;
unsigned char **data*;
int *nelements*;

XRotateWindowProperties(*display, w, properties, num_prop, npositions*)
Display **display*;
Window *w*;
Atom *properties*[];
int *num_prop*;
int *npositions*;

XDeleteProperty(*display, w, property*)
Display **display*;
Window *w*;
Atom *property*;

ARGUMENTS

actual_format_return
Returns the actual format of the property.

actual_type_return
Returns the atom identifier that defines the actual type of the property.

bytes_after_return
Returns the number of bytes remaining to be read in the property if a partial read was performed.

data Specifies the property data.

delete Specifies a Boolean value that determines whether the property is deleted.

display Specifies the connection to the X server.

format Specifies whether the data should be viewed as a list of 8-bit, 16-bit, or 32-bit quantities. Possible values are 8, 16, and 32. This information allows the X server to correctly perform byte-swap operations as necessary. If the format is 16-bit or 32-bit, you must explicitly cast your data pointer to a (char *) in the call to **XChangeProperty**.

long_length Specifies the length in 32-bit multiples of the data to be retrieved.

long_offset Specifies the offset in the specified property (in 32-bit quantities) where the data is to be retrieved.

mode Specifies the mode of the operation. You can pass **PropModeReplace**, **PropModePrepend**, or **PropModeAppend**.

nelements Specifies the number of elements of the specified data format.

nitems_return Returns the actual number of 8-bit, 16-bit, or 32-bit items stored in the prop_return data.

num_prop Specifies the length of the properties array.

num_prop_return Returns the length of the properties array.

npositions Specifies the rotation amount.

prop_return Returns a pointer to the data in the specified format.

property Specifies the property name.

properties Specifies the array of properties that are to be rotated.

req_type Specifies the atom identifier associated with the property type or **AnyPropertyType**.

type Specifies the type of the property. The X server does not interpret the type but simply passes it back to an application that later calls **XGetWindowProperty**.

w Specifies the window whose property you want to obtain, change, rotate or delete.

DESCRIPTION

The **XGetWindowProperty** function returns the actual type of the property; the actual format of the property; the number of 8-bit, 16-bit, or 32-bit items transferred; the number of bytes remaining to be read in the property; and a pointer to the data actually returned. **XGetWindowProperty** sets the return arguments as follows:

- If the specified property does not exist for the specified window, **XGetWindowProperty** returns **None** to actual_type_return and the value zero to actual_format_return and bytes_after_return. The nitems_return argument is empty. In this case, the delete argument is ignored.
- If the specified property exists but its type does not match the specified type, **XGetWindowProperty** returns the actual property type to actual_type_return, the actual property format (never zero) to actual_format_return, and the property length in bytes (even if the actual_format_return is 16 or 32) to bytes_after_return. It also ignores the delete argument. The nitems_return argument is empty.
- If the specified property exists and either you assign **AnyPropertyType** to the req_type argument or the specified type matches the actual property type, **XGetWindowProperty** returns the actual property type to actual_type_return and the actual property format

(never zero) to actual_format_return. It also returns a value to bytes_after_return and nitems_return, by defining the following values:

N = actual length of the stored property in bytes
(even if the format is 16 or 32)
I = 4 * long_offset
T = N - I
L = MINIMUM(T, 4 * long_length)
A = N - (I + L)

The returned value starts at byte index I in the property (indexing from zero), and its length in bytes is L. If the value for long_offset causes L to be negative, a **BadValue** error results. The value of bytes_after_return is A, giving the number of trailing unread bytes in the stored property.

XGetWindowProperty always allocates one extra byte in prop_return (even if the property is zero length) and sets it to ASCII null so that simple properties consisting of characters do not have to be copied into yet another string before use. If delete is **True** and bytes_after_return is zero, **XGetWindowProperty** deletes the property from the window and generates a **PropertyNotify** event on the window.

The function returns **Success** if it executes successfully. To free the resulting data, use **XFree**.

XGetWindowProperty can generate **BadAtom**, **BadValue**, and **BadWindow** errors.

The **XListProperties** function returns a pointer to an array of atom properties that are defined for the specified window or returns NULL if no properties were found. To free the memory allocated by this function, use **XFree**.

XListProperties can generate a **BadWindow** error.

The **XChangeProperty** function alters the property for the specified window and causes the X server to generate a **PropertyNotify** event on that window. **XChangeProperty** performs the following:

- If mode is **PropModeReplace**, **XChangeProperty** discards the previous property value and stores the new data.
- If mode is **PropModePrepend** or **PropModeAppend**, **XChangeProperty** inserts the specified data before the beginning of the existing data or onto the end of the existing data, respectively. The type and format must match the existing property value, or a **BadMatch** error results. If the property is undefined, it is treated as defined with the correct type and format with zero-length data.

The lifetime of a property is not tied to the storing client. Properties remain until explicitly deleted, until the window is destroyed, or until the server resets. For a discussion of what happens when the connection to the X server is closed, see section 2.5. The maximum size of a property is server dependent and can vary dynamically depending on the amount of memory the server has available. (If there is insufficient space, a **BadAlloc** error results.)

XChangeProperty can generate **BadAlloc**, **BadAtom**, **BadMatch**, **BadValue**, and **BadWindow** errors.

The **XRotateWindowProperties** function allows you to rotate properties on a window and causes the X server to generate **PropertyNotify** events. If the property names in the properties array are viewed as being numbered starting from zero and if there are num_prop property names in the list, then the value associated with property name I becomes the value associated with property name (I + npositions) mod N for all I from zero to N – 1. The effect is to rotate the states by npositions places around the virtual ring of property names (right for positive npositions, left for negative npositions). If npositions mod N is nonzero, the X server generates a

PropertyNotify event for each property in the order that they are listed in the array. If an atom occurs more than once in the list or no property with that name is defined for the window, a **BadMatch** error results. If a **BadAtom** or **BadMatch** error results, no properties are changed.

XRotateWindowProperties can generate **BadAtom**, **BadMatch**, and **BadWindow** errors.

The **XDeleteProperty** function deletes the specified property only if the property was defined on the specified window and causes the X server to generate a **PropertyNotify** event on the window unless the property does not exist.

XDeleteProperty can generate **BadAtom** and **BadWindow** errors.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadAtom A value for an Atom argument does not name a defined Atom.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XFree(3), XInternAtom(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGrabButton, XUngrabButton – grab pointer buttons

SYNTAX

XGrabButton(*display, button, modifiers, grab_window, owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor*)

Display **display*;
unsigned int *button*;
unsigned int *modifiers*;
Window *grab_window*;
Bool *owner_events*;
unsigned int *event_mask*;
int *pointer_mode*, *keyboard_mode*;
Window *confine_to*;
Cursor *cursor*;

XUngrabButton(*display, button, modifiers, grab_window*)

Display **display*;
unsigned int *button*;
unsigned int *modifiers*;
Window *grab_window*;

ARGUMENTS

button	Specifies the pointer button that is to be grabbed or released or **AnyButton**.
confine_to	Specifies the window to confine the pointer in or **None**.
cursor	Specifies the cursor that is to be displayed or **None**.
display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass **GrabModeSync** or **GrabModeAsync**.
modifiers	Specifies the set of keymasks or **AnyModifier**. The mask is the bitwise inclusive OR of the valid keymask bits.
owner_events	Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
pointer_mode	Specifies further processing of pointer events. You can pass **GrabModeSync** or **GrabModeAsync**.

DESCRIPTION

The **XGrabButton** function establishes a passive grab. In the future, the pointer is actively grabbed (as for **XGrabPointer**), the last-pointer-grab time is set to the time at which the button was pressed (as transmitted in the **ButtonPress** event), and the **ButtonPress** event is reported if all of the following conditions are true:

- The pointer is not grabbed, and the specified button is logically pressed when the specified modifier keys are logically down, and no other buttons or modifier keys are logically down.
- The grab_window contains the pointer.
- The confine_to window (if any) is viewable.
- A passive grab on the same button/key combination does not exist on any ancestor of grab_window.

The interpretation of the remaining arguments is as for **XGrabPointer**. The active grab is terminated automatically when the logical state of the pointer has all buttons released (independent of the state of the logical modifier keys).

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

This request overrides all previous grabs by the same client on the same button/key combinations on the same window. A modifiers of **AnyModifier** is equivalent to issuing the grab request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned KeyCodes. A button of **AnyButton** is equivalent to issuing the request for all possible buttons. Otherwise, it is not required that the specified button currently be assigned to a physical button.

If some other client has already issued a **XGrabButton** with the same button/key combination on the same window, a **BadAccess** error results. When using **AnyModifier** or **AnyButton**, the request fails completely, and a **BadAccess** error results (no grabs are established) if there is a conflicting grab for any combination. **XGrabButton** has no effect on an active grab.

XGrabButton can generate **BadCursor**, **BadValue**, and **BadWindow** errors.

The **XUngrabButton** function releases the passive button/key combination on the specified window if it was grabbed by this client. A modifiers of **AnyModifier** is equivalent to issuing the ungrab request for all possible modifier combinations, including the combination of no modifiers. A button of **AnyButton** is equivalent to issuing the request for all possible buttons. **XUngrabButton** has no effect on an active grab.

XUngrabButton can generate **BadValue** and **BadWindow** errors.

DIAGNOSTICS

BadCursor A value for a Cursor argument does not name a defined Cursor.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllowEvents(3), XGrabPointer(3), XGrabKey(3), XGrabKeyboard(3),
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGrabKey, XUngrabKey – grab keyboard keys

SYNTAX

XGrabKey(*display, keycode, modifiers, grab_window, owner_events, pointer_mode, keyboard_mode*)
Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *grab_window*;
Bool *owner_events*;
int *pointer_mode, keyboard_mode*;

XUngrabKey(*display, keycode, modifiers, grab_window*)
Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *grab_window*;

ARGUMENTS

display	Specifies the connection to the X server.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass **GrabModeSync** or **GrabModeAsync**.
keycode	Specifies the KeyCode or **AnyKey**.
modifiers	Specifies the set of keymasks or **AnyModifier**. The mask is the bitwise inclusive OR of the valid keymask bits.
owner_events	Specifies a Boolean value that indicates whether the keyboard events are to be reported as usual.
pointer_mode	Specifies further processing of pointer events. You can pass **GrabModeSync** or **GrabModeAsync**.

DESCRIPTION

The **XGrabKey** function establishes a passive grab on the keyboard. In the future, the keyboard is actively grabbed (as for **XGrabKeyboard**), the last-keyboard-grab time is set to the time at which the key was pressed (as transmitted in the **KeyPress** event), and the **KeyPress** event is reported if all of the following conditions are true:

- The keyboard is not grabbed and the specified key (which can itself be a modifier key) is logically pressed when the specified modifier keys are logically down, and no other modifier keys are logically down.
- Either the grab_window is an ancestor of (or is) the focus window, or the grab_window is a descendant of the focus window and contains the pointer.
- A passive grab on the same key combination does not exist on any ancestor of grab_window.

The interpretation of the remaining arguments is as for **XGrabKeyboard**. The active grab is terminated automatically when the logical state of the keyboard has the specified key released (independent of the logical state of the modifier keys).

Note that the logical state of a device (as seen by client applications) may lag the physical state if device event processing is frozen.

A modifiers argument of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). It is not required that all modifiers specified have currently assigned KeyCodes. A keycode argument of **AnyKey** is equivalent to issuing the request for all possible KeyCodes. Otherwise, the specified keycode must be in the range specified by min_keycode and max_keycode in the connection setup, or a **BadValue** error results.

If some other client has issued a **XGrabKey** with the same key combination on the same window, a **BadAccess** error results. When using **AnyModifier** or **AnyKey**, the request fails completely, and a **BadAccess** error results (no grabs are established) if there is a conflicting grab for any combination.

XGrabKey can generate **BadAccess**, **BadValue**, and **BadWindow** errors.

The **XUngrabKey** function releases the key combination on the specified window if it was grabbed by this client. It has no effect on an active grab. A modifiers of **AnyModifier** is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A keycode argument of **AnyKey** is equivalent to issuing the request for all possible key codes.

XUngrabKey can generate **BadValue** and **BadWindow** error.

DIAGNOSTICS

BadAccess A client attempted to grab a key/button combination already grabbed by another client.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllowAccess(3), XGrabButton(3), XGrabKeyboard(3), XGrabPointer(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGrabKeyboard, XUngrabKeyboard – grab the keyboard

SYNTAX

int XGrabKeyboard(*display, grab_window, owner_events, pointer_mode, keyboard_mode, time*)
Display **display*;
Window *grab_window*;
Bool *owner_events*;
int *pointer_mode, keyboard_mode*;
Time *time*;

XUngrabKeyboard(*display, time*)
Display **display*;
Time *time*;

ARGUMENTS

display	Specifies the connection to the X server.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass **GrabModeSync** or **GrabModeAsync**.
owner_events	Specifies a Boolean value that indicates whether the keyboard events are to be reported as usual.
pointer_mode	Specifies further processing of pointer events. You can pass **GrabModeSync** or **GrabModeAsync**.
time	Specifies the time. You can pass either a timestamp or **CurrentTime**.

DESCRIPTION

The **XGrabKeyboard** function actively grabs control of the keyboard and generates **FocusIn** and **FocusOut** events. Further key events are reported only to the grabbing client. **XGrabKeyboard** overrides any active keyboard grab by this client. If owner_events is **False**, all generated key events are reported with respect to grab_window. If owner_events is **True** and if a generated key event would normally be reported to this client, it is reported normally; otherwise, the event is reported with respect to the grab_window. Both **KeyPress** and **KeyRelease** events are always reported, independent of any event selection made by the client.

If the keyboard_mode argument is **GrabModeAsync**, keyboard event processing continues as usual. If the keyboard is currently frozen by this client, then processing of keyboard events is resumed. If the keyboard_mode argument is **GrabModeSync**, the state of the keyboard (as seen by client applications) appears to freeze, and the X server generates no further keyboard events until the grabbing client issues a releasing **XAllowEvents** call or until the keyboard grab is released. Actual keyboard changes are not lost while the keyboard is frozen; they are simply queued in the server for later processing.

If pointer_mode is **GrabModeAsync**, pointer event processing is unaffected by activation of the grab. If pointer_mode is **GrabModeSync**, the state of the pointer (as seen by client applications) appears to freeze, and the X server generates no further pointer events until the grabbing client issues a releasing **XAllowEvents** call or until the keyboard grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If the keyboard is actively grabbed by some other client, **XGrabKeyboard** fails and returns **AlreadyGrabbed**. If grab_window is not viewable, it fails and returns **GrabNotViewable**. If the keyboard is frozen by an active grab of another client, it fails and returns **GrabFrozen**. If the specified time is earlier than the last-keyboard-grab time or later than the current X server time, it fails and returns **GrabInvalidTime**. Otherwise, the last-keyboard-grab time is set to the specified time (**CurrentTime** is replaced by the current X server time).

XGrabKeyboard can generate **BadValue** and **BadWindow** errors.

The **XUngrabKeyboard** function releases the keyboard and any queued events if this client has it actively grabbed from either **XGrabKeyboard** or **XGrabKey**. **XUngrabKeyboard** does not release the keyboard and any queued events if the specified time is earlier than the last-keyboard-grab time or is later than the current X server time. It also generates **FocusIn** and **FocusOut** events. The X server automatically performs an **UngrabKeyboard** request if the event window for an active keyboard grab becomes not viewable.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllowEvents(3), XGrabButton(3), XGrabKey(3), XGrabPointer(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGrabPointer, XUngrabPointer, XChangeActivePointerGrab – grab the pointer

SYNTAX

int XGrabPointer(*display, grab_window, owner_events, event_mask, pointer_mode, keyboard_mode, confine_to, cursor, time*)
Display **display*;
Window *grab_window*;
Bool *owner_events*;
unsigned int *event_mask*;
int *pointer_mode, keyboard_mode*;
Window *confine_to*;
Cursor *cursor*;
Time *time*;

XUngrabPointer(*display, time*)
Display **display*;
Time *time*;

XChangeActivePointerGrab(*display, event_mask, cursor, time*)
Display **display*;
unsigned int *event_mask*;
Cursor *cursor*;
Time *time*;

ARGUMENTS

confine_to	Specifies the window to confine the pointer in or **None**.
cursor	Specifies the cursor that is to be displayed during the grab or **None**.
display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. The mask is the bitwise inclusive OR of the valid pointer event mask bits.
grab_window	Specifies the grab window.
keyboard_mode	Specifies further processing of keyboard events. You can pass **GrabModeSync** or **GrabModeAsync**.
owner_events	Specifies a Boolean value that indicates whether the pointer events are to be reported as usual or reported with respect to the grab window if selected by the event mask.
pointer_mode	Specifies further processing of pointer events. You can pass **GrabModeSync** or **GrabModeAsync**.
time	Specifies the time. You can pass either a timestamp or **CurrentTime**.

DESCRIPTION

The **XGrabPointer** function actively grabs control of the pointer and returns **GrabSuccess** if the grab was successful. Further pointer events are reported only to the grabbing client. **XGrabPointer** overrides any active pointer grab by this client. If owner_events is **False**, all generated pointer events are reported with respect to grab_window and are reported only if selected by event_mask. If owner_events is **True** and if a generated pointer event would normally be reported to this client, it is reported as usual. Otherwise, the event is reported with respect to the grab_window and is reported only if selected by event_mask. For either value of owner_events, unreported events are discarded.

If the pointer_mode is **GrabModeAsync**, pointer event processing continues as usual. If the pointer is currently frozen by this client, the processing of events for the pointer is resumed. If the pointer_mode is **GrabModeSync**, the state of the pointer, as seen by client applications,

appears to freeze, and the X server generates no further pointer events until the grabbing client calls **XAllowEvents** or until the pointer grab is released. Actual pointer changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If the keyboard_mode is **GrabModeAsync**, keyboard event processing is unaffected by activation of the grab. If the keyboard_mode is **GrabModeSync**, the state of the keyboard, as seen by client applications, appears to freeze, and the X server generates no further keyboard events until the grabbing client calls **XAllowEvents** or until the pointer grab is released. Actual keyboard changes are not lost while the pointer is frozen; they are simply queued in the server for later processing.

If a cursor is specified, it is displayed regardless of what window the pointer is in. If **None** is specified, the normal cursor for that window is displayed when the pointer is in grab_window or one of its subwindows; otherwise, the cursor for grab_window is displayed.

If a confine_to window is specified, the pointer is restricted to stay contained in that window. The confine_to window need have no relationship to the grab_window. If the pointer is not initially in the confine_to window, it is warped automatically to the closest edge just before the grab activates and enter/leave events are generated as usual. If the confine_to window is subsequently reconfigured, the pointer is warped automatically, as necessary, to keep it contained in the window.

The time argument allows you to avoid certain circumstances that come up if applications take a long time to respond or if there are long network delays. Consider a situation where you have two applications, both of which normally grab the pointer when clicked on. If both applications specify the timestamp from the event, the second application may wake up faster and successfully grab the pointer before the first application. The first application then will get an indication that the other application grabbed the pointer before its request was processed.

XGrabPointer generates **EnterNotify** and **LeaveNotify** events.

Either if grab_window or confine_to window is not viewable or if the confine_to window lies completely outside the boundaries of the root window, **XGrabPointer** fails and returns **GrabNotViewable**. If the pointer is actively grabbed by some other client, it fails and returns **AlreadyGrabbed**. If the pointer is frozen by an active grab of another client, it fails and returns **GrabFrozen**. If the specified time is earlier than the last-pointer-grab time or later than the current X server time, it fails and returns **GrabInvalidTime**. Otherwise, the last-pointer-grab time is set to the specified time (**CurrentTime** is replaced by the current X server time).

XGrabPointer can generate **BadCursor**, **BadValue**, and **BadWindow** errors.

The **XUngrabPointer** function releases the pointer and any queued events if this client has actively grabbed the pointer from **XGrabPointer**, **XGrabButton**, or from a normal button press. **XUngrabPointer** does not release the pointer if the specified time is earlier than the last-pointer-grab time or is later than the current X server time. It also generates **EnterNotify** and **LeaveNotify** events. The X server performs an **UngrabPointer** request automatically if the event window or confine_to window for an active pointer grab becomes not viewable or if window reconfiguration causes the confine_to window to lie completely outside the boundaries of the root window.

The **XChangeActivePointerGrab** function changes the specified dynamic parameters if the pointer is actively grabbed by the client and if the specified time is no earlier than the last-pointer-grab time and no later than the current X server time. This function has no effect on the passive parameters of a **XGrabButton**. The interpretation of event_mask and cursor is the same as described in **XGrabPointer**.

XChangeActivePointerGrab can generate a **BadCursor** and **BadValue** error.

DIAGNOSTICS

BadCursor A value for a Cursor argument does not name a defined Cursor.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllowEvents(3), XGrabButton(3), XGrabKey(3), XGrabKeyboard(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGrabServer, XUngrabServer – grab the server

SYNTAX

XGrabServer(*display*)
Display ** *display*;

XUngrabServer(*display*)
Display ** *display*;

ARGUMENTS

display Specifies the connection to the X server.

DESCRIPTION

The **XGrabServer** function disables processing of requests and close downs on all other connections than the one this request arrived on. You should not grab the X server any more than is absolutely necessary.

The **XUngrabServer** function restarts processing of requests and close downs on other connections. You should avoid grabbing the X server as much as possible.

SEE ALSO

XGrabButton(3), XGrabKey(3), XGrabKeyboard(3), XGrabPointer(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGraphicsExposeEvent, XNoExposeEvent – GraphicsExpose and NoExpose event structures

STRUCTURES

The structures for **GraphicsExpose** and **NoExpose** events contain:

```
typedef struct {
        int type;                       /* GraphicsExpose */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Drawable drawable;
        int x, y;
        int width, height;
        int count;                      /* if nonzero, at least this many more */
        int major_code;                 /* core is CopyArea or CopyPlane */
        int minor_code;                 /* not defined in the core */
} XGraphicsExposeEvent;

typedef struct {
        int type;                       /* NoExpose */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Drawable drawable;
        int major_code;                 /* core is CopyArea or CopyPlane */
        int minor_code;                 /* not defined in the core */
} XNoExposeEvent;
```

When you receive these events, their structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

Both structures have these common members: drawable, major_code, and minor_code. The drawable member is set to the drawable of the destination region on which the graphics request was to be performed. The major_code member is set to the graphics request initiated by the client and can be either **X_CopyArea** or **X_CopyPlane**. If it is **X_CopyArea**, a call to **XCopyArea** initiated the request. If it is **X_CopyPlane**, a call to **XCopyPlane** initiated the request. These constants are defined in <**X11/Xproto.h**>. The minor_code member, like the major_code member, indicates which graphics request was initiated by the client. However, the minor_code member is not defined by the core X protocol and will be zero in these cases, although it may be used by an extension.

The **XGraphicsExposeEvent** structure has these additional members: x, y, width, height, and count. The x and y members are set to the coordinates relative to the drawable's origin and indicate the upper-left corner of the rectangle. The width and height members are set to the size (extent) of the rectangle. The count member is set to the number of **GraphicsExpose** events to follow. If count is zero, no more **GraphicsExpose** events follow for this window. However, if count is nonzero, at least that number of **GraphicsExpose** events (and possibly more) are to follow for this window.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCopyArea(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XGravityEvent – GravityNotify event structure

STRUCTURES

The structure for **GravityNotify** events contains:

```
typedef struct {
        int type;                       /* GravityNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
        int x, y;
} XGravityEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the window that was moved or to its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. The window member is set to the child window that was moved. The x and y members are set to the coordinates relative to the new parent window's origin and indicate the position of the upper-left outside corner of the window.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XIconifyWindow, XWithdrawWindow, XReconfigureWMWindow - manipulate top-level windows

SYNTAX

Status XIconifyWindow(*display, w, screen_number*)
Display **display*;
Window *w*;
int *screen_number*;

Status XWithdrawWindow(*display, w, screen_number*)
Display **display*;
Window *w*;
int *screen_number*;

Status XReconfigureWMWindow(*display, w, screen_number, value_mask, values*)
Display **display*;
Window *w*;
int *screen_number*;
unsigned int *value_mask*;
XWindowChanges **values*;

ARGUMENTS

display	Specifies the connection to the X server.
screen_number	Specifies the appropriate screen number on the host server.
value_mask	Specifies which values are to be set using information in the values structure. This mask is the bitwise inclusive OR of the valid configure window values bits.
values	Specifies a pointer to the **XWindowChanges** structure.
w	Specifies the window.

DESCRIPTION

The **XIconifyWindow** function sends a WM_CHANGE_STATE **ClientMessage** event with a format of 32 and a first data element of **IconicState** (as described in Section 4.1.4 of the *Inter-Client Communication Conventions Manual*) to the root window of the specified screen. Window managers may elect to receive this message and, if the window is in its normal state, may treat it as a request to change the window's state from normal to iconic. If the WM_CHANGE_STATE property cannot be interned, **XIconifyWindow** does not send a message and returns a zero status. It returns a nonzero status if the client message is sent successfully; otherwise, it returns a zero status.

XIconifyWindow can generate a **BadWindow** error.

The **XWithdrawWindow** function unmaps the specified window and sends a synthetic **UnmapNotify** event to the root window of the specified screen. Window managers may elect to receive this message and may treat it as a request to change the window's state to withdrawn. When a window is in the withdrawn state, neither its normal nor its iconic representations is visible. It returns a nonzero status if the **UnmapNotify** event is successfully sent; otherwise, it returns a zero status.

XWithdrawWindow can generate a **BadWindow** error.

The **XReconfigureWMWindow** function issues a **ConfigureWindow** request on the specified top-level window. If the stacking mode is changed and the request fails with a **BadMatch** error, the error event is trapped and a synthetic **ConfigureRequestEvent** containing the same configuration parameters is sent to the root of the specified window. Window managers may elect to receive this event and treat it as a request to reconfigure the indicated window.

XReconfigureWMWindow can generate **BadValue** and **BadWindow** errors.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XConfigureWindow(3), XCreateWindow(3), XDestroyWindow(3), XRaiseWindow(3), XMapWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XIfEvent, XCheckIfEvent, XPeekIfEvent – check the event queue with a predicate procedure

SYNTAX

XIfEvent(*display, event_return, predicate, arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

Bool XCheckIfEvent(*display, event_return, predicate, arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

XPeekIfEvent(*display, event_return, predicate, arg*)
Display **display*;
XEvent **event_return*;
Bool (**predicate*)();
char **arg*;

ARGUMENTS

arg	Specifies the user-supplied argument that will be passed to the predicate procedure.
display	Specifies the connection to the X server.
event_return	Returns either a copy of or the matched event's associated structure.
predicate	Specifies the procedure that is to be called to determine if the next event in the queue matches what you want.

DESCRIPTION

The **XIfEvent** function completes only when the specified predicate procedure returns **True** for an event, which indicates an event in the queue matches. **XIfEvent** flushes the output buffer if it blocks waiting for additional events. **XIfEvent** removes the matching event from the queue and copies the structure into the client-supplied **XEvent** structure.

When the predicate procedure finds a match, **XCheckIfEvent** copies the matched event into the client-supplied **XEvent** structure and returns **True**. (This event is removed from the queue.) If the predicate procedure finds no match, **XCheckIfEvent** returns **False**, and the output buffer will have been flushed. All earlier events stored in the queue are not discarded.

The **XPeekIfEvent** function returns only when the specified predicate procedure returns **True** for an event. After the predicate procedure finds a match, **XPeekIfEvent** copies the matched event into the client-supplied **XEvent** structure without removing the event from the queue. **XPeekIfEvent** flushes the output buffer if it blocks waiting for additional events.

SEE ALSO

XAnyEvent(3), XNextEvent(3), XPutBackEvent(3) XSendEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XInstallColormap, XUninstallColormap, XListInstalledColormaps - control colormaps

SYNTAX

XInstallColormap(*display, colormap*)
Display **display*;
Colormap *colormap*;

XUninstallColormap(*display, colormap*)
Display **display*;
Colormap *colormap*;

Colormap *XListInstalledColormaps(*display, w, num_return*)
Display **display*;
Window *w*;
int **num_return*;

ARGUMENTS

colormap	Specifies the colormap.
display	Specifies the connection to the X server.
num_return	Returns the number of currently installed colormaps.
w	Specifies the window that determines the screen.

DESCRIPTION

The **XInstallColormap** function installs the specified colormap for its associated screen. All windows associated with this colormap immediately display with true colors. You associated the windows with this colormap when you created them by calling **XCreateWindow**, **XCreateSimpleWindow**, **XChangeWindowAttributes**, or **XSetWindowColormap**.

If the specified colormap is not already an installed colormap, the X server generates a **ColormapNotify** event on each window that has that colormap. In addition, for every other colormap that is installed as a result of a call to **XInstallColormap**, the X server generates a **ColormapNotify** event on each window that has that colormap.

XInstallColormap can generate a **BadColor** error.

The **XUninstallColormap** function removes the specified colormap from the required list for its screen. As a result, the specified colormap might be uninstalled, and the X server might implicitly install or uninstall additional colormaps. Which colormaps get installed or uninstalled is server-dependent except that the required list must remain installed.

If the specified colormap becomes uninstalled, the X server generates a **ColormapNotify** event on each window that has that colormap. In addition, for every other colormap that is installed or uninstalled as a result of a call to **XUninstallColormap**, the X server generates a **ColormapNotify** event on each window that has that colormap.

XUninstallColormap can generate a **BadColor** error.

The **XListInstalledColormaps** function returns a list of the currently installed colormaps for the screen of the specified window. The order of the colormaps in the list is not significant and is no explicit indication of the required list. When the allocated list is no longer needed, free it by using **XFree**.

XListInstalledColormaps can generate a **BadWindow** error.

DIAGNOSTICS

BadColor	A value for a Colormap argument does not name a defined Colormap.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAtrributes(3), XCreateColormap(3), XCreateWindow(3), XFree(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XInternAtom, XGetAtomName – create or return atom names

SYNTAX

Atom XInternAtom(*display, atom_name, only_if_exists*)
Display **display*;
char **atom_name*;
Bool *only_if_exists*;

char *XGetAtomName(*display, atom*)
Display **display*;
Atom *atom*;

ARGUMENTS

atom	Specifies the atom for the property name you want returned.
atom_name	Specifies the name associated with the atom you want returned.
display	Specifies the connection to the X server.
only_if_exists	Specifies a Boolean value that indicates whether **XInternAtom** creates the atom.

DESCRIPTION

The **XInternAtom** function returns the atom identifier associated with the specified atom_name string. If only_if_exists is **False**, the atom is created if it does not exist. Therefore, **XInternAtom** can return **None**. You should use a null-terminated ISO Latin-1 string for atom_name. Case matters; the strings *thing*, *Thing*, and *thinG* all designate different atoms. The atom will remain defined even after the client's connection closes. It will become undefined only when the last connection to the X server closes.

XInternAtom can generate **BadAlloc** and **BadValue** errors.

The **XGetAtomName** function returns the name associated with the specified atom. To free the resulting string, call **XFree**.

XGetAtomName can generate a **BadAtom** error.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XFree(3), XGetWindowProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XIntersectRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XXorRegion, XOffsetRegion, XShrinkRegion – region arthmetic

SYNTAX

XIntersectRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XUnionRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XUnionRectWithRegion(*rectangle*, *src_region*, *dest_region_return*)
XRectangle **rectangle*;
Region *src_region*;
Region *dest_region_return*;

XSubtractRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XXorRegion(*sra*, *srb*, *dr_return*)
Region *sra*, *srb*, *dr_return*;

XOffsetRegion(*r*, *dx*, *dy*)
Region *r*;
int *dx*, *dy*;

XShrinkRegion(*r*, *dx*, *dy*)
Region *r*;
int *dx*, *dy*;

ARGUMENTS

dest_region_return Returns the destination region.

dr_return Returns the result of the computation. ds Dy move or shrink

dx
dy Specify the x and y coordinates, which define the amount you want to the specified region.

r Specifies the region.

rectangle Specifies the rectangle.

sra
srb Specify the two regions with which you want to perform the computation.

src_region Specifies the source region to be used.

DESCRIPTION

The **XIntersectRegion** function computes the intersection of two regions.

The **XUnionRegion** function computes the union of two regions.

The **XUnionRectWithRegion** function updates the destination region from a union of the specified rectangle and the specified source region.

The **XSubtractRegion** function subtracts srb from sra and stores the results in dr_return.

The **XXorRegion** function calculates the difference between the union and intersection of two regions.

The **XOffsetRegion** function moves the specified region by a specified amount.

The **XShrinkRegion** function reduces the specified region by a specified amount. Positive values shrink the size of the region, and negative values expand the region.

SEE ALSO

XCreateRegion(3), XDrawRectangle(3), XEmptyRegion(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XKeymapEvent - KeymapNotify event structure

STRUCTURES

The structure for **KeymapNotify** events contains:

```
/* generated on EnterWindow and FocusIn when KeymapState selected */
typedef struct {
        int type;                       /* KeymapNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        char key_vector[32];
} XKeymapEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is not used but is present to aid some toolkits. The key_vector member is set to the bit vector of the keyboard. Each bit set to 1 indicates that the corresponding key is currently pressed. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N + 7 with the least-significant bit in the byte representing key 8N.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XListFonts, XFreeFontNames, XListFontsWithInfo, XFreeFontInfo – obtain or free font names and information

SYNTAX

char **XListFonts(*display, pattern, maxnames, actual_count_return*)
Display *display;
char *pattern;
int maxnames;
int *actual_count_return;

XFreeFontNames(*list*)
char *list[];

char **XListFontsWithInfo(*display, pattern, maxnames, count_return, info_return*)
Display *display;
char *pattern;
int maxnames;
int *count_return;
XFontStruct **info_return;

XFreeFontInfo(*names, free_info, actual_count*)
char **names;
XFontStruct *free_info;
int actual_count;

ARGUMENTS

actual_count Specifies the actual number of matched font names returned by **XListFontsWithInfo**.

actual_count_return
Returns the actual number of font names.

count_return Returns the actual number of matched font names.

display Specifies the connection to the X server.

info_return Returns a pointer to the font information.

free_info Specifies the pointer to the font information returned by **XListFontsWithInfo**.

list Specifies the array of strings you want to free.

maxnames Specifies the maximum number of names to be returned.

names Specifies the list of font names returned by **XListFontsWithInfo**.

pattern Specifies the null-terminated pattern string that can contain wildcard characters.

DESCRIPTION

The **XListFonts** function returns an array of available font names (as controlled by the font search path; see **XSetFontPath**) that match the string you passed to the pattern argument. The string should be ISO Latin-1; uppercase and lowercase do not matter. Each string is terminated by an ASCII null. The pattern string can contain any characters, but each asterisk (*) is a wildcard for any number of characters, and each question mark (?) is a wildcard for a single character. The client should call **XFreeFontNames** when finished with the result to free the memory.

The **XFreeFontNames** function frees the array and strings returned by **XListFonts** or **XListFontsWithInfo**.

The **XListFontsWithInfo** function returns a list of font names that match the specified pattern and their associated font information. The list of names is limited to size specified by maxnames. The information returned for each font is identical to what **XLoadQueryFont** would return

except that the per-character metrics are not returned. The pattern string can contain any characters, but each asterisk (*) is a wildcard for any number of characters, and each question mark (?) is a wildcard for a single character. To free the allocated name array, the client should call **XFreeFontNames**. To free the the font information array, the client should call **XFreeFontInfo**.

The **XFreeFontInfo** function frees the the font information array. To free an **XFontStruct** structure without closing the font, call **XFreeFontInfo** with the names argument specified as NULL.

SEE ALSO

XLoadFont(3), XSetFontPath(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XLoadFont, XQueryFont, XLoadQueryFont, XFreeFont, XGetFontProperty, XUnloadFont, XCharStruct, XFontProp, XChar2b, XFontStruct – load or unload fonts and font metric structures

SYNTAX

Font XLoadFont(*display, name*)
Display **display*;
char **name*;

XFontStruct *XQueryFont(*display, font_ID*)
Display **display*;
XID *font_ID*;

XFontStruct *XLoadQueryFont(*display, name*)
Display **display*;
char **name*;

XFreeFont(*display, font_struct*)
Display **display*;
XFontStruct **font_struct*;

Bool XGetFontProperty(*font_struct, atom, value_return*)
XFontStruct **font_struct*;
Atom *atom*;
unsigned long **value_return*;

XUnloadFont(*display, font*)
Display **display*;
Font *font*;

ARGUMENTS

atom	Specifies the atom for the property name you want returned.
display	Specifies the connection to the X server.
font	Specifies the font.
font_ID	Specifies the font ID or the **GContext** ID.
font_struct	Specifies the storage associated with the font.
gc	Specifies the GC.
name	Specifies the name of the font, which is a null-terminated string.
value_return	Returns the value of the font property.

DESCRIPTION

The **XLoadFont** function loads the specified font and returns its associated font ID. The name should be ISO Latin-1 encoding; uppercase and lowercase do not matter. The interpretation of characters "?" (octal value 77) and "*" (octal value 52) in the name is not defined by the core protocol but is reserved for future definition. A structured format for font names is specified in the X Consortium standard *X Logical Font Description Conventions*. If **XLoadFont** was unsuccessful at loading the specified font, a **BadName** error results. Fonts are not associated with a particular screen and can be stored as a component of any GC. When the font is no longer needed, call **XUnloadFont**.

XLoadFont can generate **BadAlloc** and **BadName** errors.

The **XQueryFont** function returns a pointer to the **XFontStruct** structure, which contains information associated with the font. You can query a font or the font stored in a GC. The font ID stored in the **XFontStruct** structure will be the **GContext** ID, and you need to be careful when using this ID in other functions (see **XGContextFromGC**). To free this data, use

XFreeFontInfo.

XLoadQueryFont can generate a **BadAlloc** error.

The **XLoadQueryFont** function provides the most common way for accessing a font. **XLoadQueryFont** both opens (loads) the specified font and returns a pointer to the appropriate **XFontStruct** structure. If the font does not exist, **XLoadQueryFont** returns NULL.

The **XFreeFont** function deletes the association between the font resource ID and the specified font and frees the **XFontStruct** structure. The font itself will be freed when no other resource references it. The data and the font should not be referenced again.

XFreeFont can generate a **BadFont** error.

Given the atom for that property, the **XGetFontProperty** function returns the value of the specified font property. **XGetFontProperty** also returns **False** if the property was not defined or **True** if it was defined. A set of predefined atoms exists for font properties, which can be found in <**X11/Xatom.h**>. This set contains the standard properties associated with a font. Although it is not guaranteed, it is likely that the predefined font properties will be present.

The **XUnloadFont** function deletes the association between the font resource ID and the specified font. The font itself will be freed when no other resource references it. The font should not be referenced again.

XUnloadFont can generate a **BadFont** error.

STRUCTURES

The **XFontStruct** structure contains all of the information for the font and consists of the font-specific information as well as a pointer to an array of **XCharStruct** structures for the characters contained in the font. The **XFontStruct**, **XFontProp**, and **XCharStruct** structures contain:

```
typedef struct {
        short lbearing;                 /* origin to left edge of raster */
        short rbearing;                 /* origin to right edge of raster */
        short width;                    /* advance to next char's origin */
        short ascent;                   /* baseline to top edge of raster */
        short descent;                  /* baseline to bottom edge of raster */
        unsigned short attributes;      /* per char flags (not predefined) */
} XCharStruct;

typedef struct {
        Atom    name;
        unsigned long card32;
} XFontProp;

typedef struct {                        /* normal 16 bit characters are two bytes */
    unsigned char byte1;
    unsigned char byte2;
} XChar2b;

typedef struct {
        XExtData *ext_data;             /* hook for extension to hang data */
        Font fid;                       /* Font id for this font */
        unsigned direction;             /* hint about the direction font is painted */
        unsigned min_char_or_byte2;     /* first character */
        unsigned max_char_or_byte2;     /* last character */
        unsigned min_byte1;             /* first row that exists */
        unsigned max_byte1;             /* last row that exists */
        Bool all_chars_exist;           /* flag if all characters have nonzero size */
        unsigned default_char;          /* char to print for undefined character */
```

```
    int n_properties;           /* how many properties there are */
    XFontProp *properties;      /* pointer to array of additional properties */
    XCharStruct min_bounds;     /* minimum bounds over all existing char */
    XCharStruct max_bounds;     /* maximum bounds over all existing char */
    XCharStruct *per_char;      /* first_char to last_char information */
    int ascent;                 /* logical extent above baseline for spacing */
    int descent;                /* logical decent below baseline for spacing */
} XFontStruct;
```

X supports single byte/character, two bytes/character matrix, and 16-bit character text operations. Note that any of these forms can be used with a font, but a single byte/character text request can only specify a single byte (that is, the first row of a 2-byte font). You should view 2-byte fonts as a two-dimensional matrix of defined characters: byte1 specifies the range of defined rows and byte2 defines the range of defined columns of the font. Single byte/character fonts have one row defined, and the byte2 range specified in the structure defines a range of characters.

The bounding box of a character is defined by the **XCharStruct** of that character. When characters are absent from a font, the default_char is used. When fonts have all characters of the same size, only the information in the **XFontStruct** min and max bounds are used.

The members of the **XFontStruct** have the following semantics:

- The direction member can be either **FontLeftToRight** or **FontRightToLeft**. It is just a hint as to whether most **XCharStruct** elements have a positive (**FontLeftToRight**) or a negative (**FontRightToLeft**) character width metric. The core protocol defines no support for vertical text.
- If the min_byte1 and max_byte1 members are both zero, min_char_or_byte2 specifies the linear character index corresponding to the first element of the per_char array, and max_char_or_byte2 specifies the linear character index of the last element.

  If either min_byte1 or max_byte1 are nonzero, both min_char_or_byte2 and max_char_or_byte2 are less than 256, and the 2-byte character index values corresponding to the per_char array element N (counting from 0) are:

  byte1 = N/D + min_byte1
  byte2 = N\D + min_char_or_byte2

  where:

  D = max_char_or_byte2 - min_char_or_byte2 + 1
  / = integer division
  \ = integer modulus
- If the per_char pointer is NULL, all glyphs between the first and last character indexes inclusive have the same information, as given by both min_bounds and max_bounds.
- If all_chars_exist is **True**, all characters in the per_char array have nonzero bounding boxes.
- The default_char member specifies the character that will be used when an undefined or nonexistent character is printed. The default_char is a 16-bit character (not a 2-byte character). For a font using 2-byte matrix format, the default_char has byte1 in the most-significant byte and byte2 in the least-significant byte. If the default_char itself specifies an undefined or nonexistent character, no printing is performed for an undefined or nonexistent character.
- The min_bounds and max_bounds members contain the most extreme values of each

individual **XCharStruct** component over all elements of this array (and ignore nonexistent characters). The bounding box of the font (the smallest rectangle enclosing the shape obtained by superimposing all of the characters at the same origin [x,y]) has its upper-left coordinate at:

[x + min_bounds.lbearing, y - max_bounds.ascent]

Its width is:

max_bounds.rbearing - min_bounds.lbearing

Its height is:

max_bounds.ascent + max_bounds.descent

- The ascent member is the logical extent of the font above the baseline that is used for determining line spacing. Specific characters may extend beyond this.
- The descent member is the logical extent of the font at or below the baseline that is used for determining line spacing. Specific characters may extend beyond this.
- If the baseline is at Y-coordinate y, the logical extent of the font is inclusive between the Y-coordinate values (y - font.ascent) and (y + font.descent - 1). Typically, the minimum interline spacing between rows of text is given by ascent + descent.

For a character origin at [x,y], the bounding box of a character (that is, the smallest rectangle that encloses the character's shape) described in terms of **XCharStruct** components is a rectangle with its upper-left corner at:

[x + lbearing, y - ascent]

Its width is:

rbearing - lbearing

Its height is:

ascent + descent

The origin for the next character is defined to be:

[x + width, y]

The lbearing member defines the extent of the left edge of the character ink from the origin. The rbearing member defines the extent of the right edge of the character ink from the origin. The ascent member defines the extent of the top edge of the character ink from the origin. The descent member defines the extent of the bottom edge of the character ink from the origin. The width member defines the logical width of the character.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadFont A value for a Font or GContext argument does not name a defined Font.

BadName A font or color of the specified name does not exist.

SEE ALSO

XCreateGC(3X11), XListFonts(3X11), XSetFontPath(3X11)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XLookupKeysym, XRefreshKeyboardMapping, XLookupString, XRebindKeySym – handle keyboard input events

SYNTAX

KeySym XLookupKeysym(*key_event, index*)
XKeyEvent **key_event*;
int *index*;

XRefreshKeyboardMapping(*event_map*)
XMappingEvent **event_map*;

int XLookupString(*event_struct, buffer_return, bytes_buffer, keysym_return, status_in_out*)
XKeyEvent **event_struct*;
char **buffer_return*;
int *bytes_buffer*;
KeySym **keysym_return*;
XComposeStatus **status_in_out*;

XRebindKeysym(*display, keysym, list, mod_count, string, bytes_string*)
Display **display*;
KeySym *keysym*;
KeySym *list*[];
int *mod_count*;
unsigned char **string*;
int *bytes_string*;

ARGUMENTS

buffer_return	Returns the translated characters.
bytes_buffer	Specifies the length of the buffer. No more than bytes_buffer of translation are returned.
bytes_string	Specifies the length of the string.
display	Specifies the connection to the X server.
event_map	Specifies the mapping event that is to be used.
event_struct	Specifies the key event structure to be used. You can pass **XKeyPressedEvent** or **XKeyReleasedEvent**.
index	Specifies the index into the KeySyms list for the event's KeyCode.
key_event	Specifies the **KeyPress** or **KeyRelease** event.
keysym	Specifies the KeySym that is to be .
keysym_return	Returns the KeySym computed from the event if this argument is not NULL.
list	Specifies the KeySyms to be used as modifiers.
mod_count	Specifies the number of modifiers in the modifier list.
status_in_out	Specifies or returns the **XComposeStatus** structure or NULL.
string	Specifies a pointer to the string that is copied and will be returned by **XLookupString**.

DESCRIPTION

The **XLookupKeysym** function uses a given keyboard event and the index you specified to return the KeySym from the list that corresponds to the KeyCode member in the **XKeyPressedEvent** or **XKeyReleasedEvent** structure. If no KeySym is defined for the KeyCode of the event, **XLookupKeysym** returns **NoSymbol**.

The **XRefreshKeyboardMapping** function refreshes the stored modifier and keymap information. You usually call this function when a **MappingNotify** event with a request member of **MappingKeyboard** or **MappingModifier** occurs. The result is to update Xlib's knowledge of the keyboard.

The **XLookupString** function translates a key event to a KeySym and a string. The KeySym is obtained by using the standard interpretation of the Shift, Lock, and group modifiers as defined in the X Protocol specification. If the KeySym has been rebound (see **XRebindKeysym**), the bound string will be stored in the buffer. Otherwise, the KeySym is mapped, if possible, to an ISO Latin-1 character or (if the Control modifier is on) to an ASCII control character, and that character is stored in the buffer. **XLookupString** returns the number of characters that are stored in the buffer.

If present (non-NULL), the **XComposeStatus** structure records the state, which is private to Xlib, that needs preservation across calls to **XLookupString** to implement compose processing.

The **XRebindKeysym** function can be used to rebind the meaning of a KeySym for the client. It does not redefine any key in the X server but merely provides an easy way for long strings to be attached to keys. **XLookupString** returns this string when the appropriate set of modifier keys are pressed and when the KeySym would have been used for the translation. Note that you can rebind a KeySym that may not exist.

SEE ALSO

XButtonEvent(3), XMapEvent(3), XStringToKeysym(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XMapEvent, XMappingEvent – MapNotify and MappingNotify event structures

STRUCTURES

The structure for **MapNotify** events contains:

```
typedef struct {
        int type;                       /* MapNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
        Bool override_redirect;         /* boolean, is override set... */
} XMapEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the window that was mapped or to its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. The window member is set to the window that was mapped. The override_redirect member is set to the override-redirect attribute of the window. Window manager clients normally should ignore this window if the override-redirect attribute is **True**, because these events usually are generated from pop-ups, which override structure control.

The structure for **MappingNotify** events is:

```
typedef struct {
        int type;                       /* MappingNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;                  /* unused */
        int request;                    /* one of MappingModifier, MappingKeyboard,
                                           MappingPointer */
        int first_keycode;              /* first keycode */
        int count;                      /* defines range of change w. first_keycode*/
} XMappingEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The request member is set to indicate the kind of mapping change that occurred and can be **MappingModifier**, **MappingKeyboard**, **MappingPointer**. If it is **MappingModifier**, the

modifier mapping was changed. If it is **MappingKeyboard**, the keyboard mapping was changed. If it is **MappingPointer**, the pointer button mapping was changed. The first_keycode and count members are set only if the request member was set to **MappingKeyboard**. The number in first_keycode represents the first number in the range of the altered mapping, and count represents the number of keycodes altered.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XMapRequestEvent – MapRequest event structure

STRUCTURES

The structure for **MapRequest** events contains:

```
typedef struct {
        int type;                       /* MapRequest */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window parent;
        Window window;
} XMapRequestEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The parent member is set to the parent window. The window member is set to the window to be mapped.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XMapWindow, XMapRaised, XMapSubwindows – map windows

SYNTAX

XMapWindow(*display*, *w*)
Display **display*;
Window *w*;

XMapRaised(*display*, *w*)
Display **display*;
Window *w*;

XMapSubwindows(*display*, *w*)
Display **display*;
Window *w*;

ARGUMENTS

display	Specifies the connection to the X server.
w	Specifies the window.

DESCRIPTION

The **XMapWindow** function maps the window and all of its subwindows that have had map requests. Mapping a window that has an unmapped ancestor does not display the window but marks it as eligible for display when the ancestor becomes mapped. Such a window is called unviewable. When all its ancestors are mapped, the window becomes viewable and will be visible on the screen if it is not obscured by another window. This function has no effect if the window is already mapped.

If the override-redirect of the window is **False** and if some other client has selected **SubstructureRedirectMask** on the parent window, then the X server generates a **MapRequest** event, and the **XMapWindow** function does not map the window. Otherwise, the window is mapped, and the X server generates a **MapNotify** event.

If the window becomes viewable and no earlier contents for it are remembered, the X server tiles the window with its background. If the window's background is undefined, the existing screen contents are not altered, and the X server generates zero or more **Expose** events. If backing-store was maintained while the window was unmapped, no **Expose** events are generated. If backing-store will now be maintained, a full-window exposure is always generated. Otherwise, only visible regions may be reported. Similar tiling and exposure take place for any newly viewable inferiors.

If the window is an **InputOutput** window, **XMapWindow** generates **Expose** events on each **InputOutput** window that it causes to be displayed. If the client maps and paints the window and if the client begins processing events, the window is painted twice. To avoid this, first ask for **Expose** events and then map the window, so the client processes input events as usual. The event list will include **Expose** for each window that has appeared on the screen. The client's normal response to an **Expose** event should be to repaint the window. This method usually leads to simpler programs and to proper interaction with window managers.

XMapWindow can generate a **BadWindow** error.

The **XMapRaised** function essentially is similar to **XMapWindow** in that it maps the window and all of its subwindows that have had map requests. However, it also raises the specified window to the top of the stack.

XMapRaised can generate a **BadWindow** error.

The **XMapSubwindows** function maps all subwindows for a specified window in top-to-bottom stacking order. The X server generates **Expose** events on each newly displayed window. This may be much more efficient than mapping many windows one at a time because the server needs

to perform much of the work only once, for all of the windows, rather than for each window.

XMapSubwindows can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XConfigureWindow(3), XCreateWindow(3), XDestroyWindow(3), XRaiseWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XNextEvent, XPeekEvent, XWindowEvent, XCheckWindowEvent, XMaskEvent, XCheck-MaskEvent, XCheckTypedEvent, XCheckTypedWindowEvent – select events by type

SYNTAX

XNextEvent(*display, event_return*)
Display * *display*;
XEvent * *event_return*;

XPeekEvent(*display, event_return*)
Display * *display*;
XEvent * *event_return*;

XWindowEvent(*display, w, event_mask, event_return*)
Display * *display*;
Window *w*;
long *event_mask*;
XEvent * *event_return*;

Bool XCheckWindowEvent(*display, w, event_mask, event_return*)
Display * *display*;
Window *w*;
long *event_mask*;
XEvent * *event_return*;

XMaskEvent(*display, event_mask, event_return*)
Display * *display*;
long *event_mask*;
XEvent * *event_return*;

Bool XCheckMaskEvent(*display, event_mask, event_return*)
Display * *display*;
long *event_mask*;
XEvent * *event_return*;

Bool XCheckTypedEvent(*display, event_type, event_return*)
Display * *display*;
int *event_type*;
XEvent * *event_return*;

Bool XCheckTypedWindowEvent(*display, w, event_type, event_return*)
Display * *display*;
Window *w*;
int *event_type*;
XEvent * *event_return*;

ARGUMENTS

display	Specifies the connection to the X server.
event_mask	Specifies the event mask.
event_return	Returns the matched event's associated structure.
event_return	Returns the next event in the queue.
event_return	Returns a copy of the matched event's associated structure.
event_type	Specifies the event type to be compared.
w	Specifies the window whose event uou are interested in.

DESCRIPTION

The **XNextEvent** function copies the first event from the event queue into the specified **XEvent** structure and then removes it from the queue. If the event queue is empty, **XNextEvent** flushes the output buffer and blocks until an event is received.

The **XPeekEvent** function returns the first event from the event queue, but it does not remove the event from the queue. If the queue is empty, **XPeekEvent** flushes the output buffer and blocks until an event is received. It then copies the event into the client-supplied **XEvent** structure without removing it from the event queue.

The **XWindowEvent** function searches the event queue for an event that matches both the specified window and event mask. When it finds a match, **XWindowEvent** removes that event from the queue and copies it into the specified **XEvent** structure. The other events stored in the queue are not discarded. If a matching event is not in the queue, **XWindowEvent** flushes the output buffer and blocks until one is received.

The **XCheckWindowEvent** function searches the event queue and then the events available on the server connection for the first event that matches the specified window and event mask. If it finds a match, **XCheckWindowEvent** removes that event, copies it into the specified **XEvent** structure, and returns **True**. The other events stored in the queue are not discarded. If the event you requested is not available, **XCheckWindowEvent** returns **False**, and the output buffer will have been flushed.

The **XMaskEvent** function searches the event queue for the events associated with the specified mask. When it finds a match, **XMaskEvent** removes that event and copies it into the specified **XEvent** structure. The other events stored in the queue are not discarded. If the event you requested is not in the queue, **XMaskEvent** flushes the output buffer and blocks until one is received.

The **XCheckMaskEvent** function searches the event queue and then any events available on the server connection for the first event that matches the specified mask. If it finds a match, **XCheckMaskEvent** removes that event, copies it into the specified **XEvent** structure, and returns **True**. The other events stored in the queue are not discarded. If the event you requested is not available, **XCheckMaskEvent** returns **False**, and the output buffer will have been flushed.

The **XCheckTypedEvent** function searches the event queue and then any events available on the server connection for the first event that matches the specified type. If it finds a match, **XCheckTypedEvent** removes that event, copies it into the specified **XEvent** structure, and returns **True**. The other events in the queue are not discarded. If the event is not available, **XCheckTypedEvent** returns **False**, and the output buffer will have been flushed.

The **XCheckTypedWindowEvent** function searches the event queue and then any events available on the server connection for the first event that matches the specified type and window. If it finds a match, **XCheckTypedWindowEvent** removes the event from the queue, copies it into the specified **XEvent** structure, and returns **True**. The other events in the queue are not discarded. If the event is not available, **XCheckTypedWindowEvent** returns **False**, and the output buffer will have been flushed.

SEE ALSO

XAnyEvent(3), XIfEvent(3), XPutBackEvent(3), XSendEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XOpenDisplay, XCloseDisplay – connect or disconnect to X server

SYNTAX

Display *XOpenDisplay(*display_name*)
char **display_name*;

XCloseDisplay(*display*)
Display **display*;

ARGUMENTS

display Specifies the connection to the X server.

display_name Specifies the hardware display name, which determines the display and communications domain to be used. On a POSIX system, if the display_name is NULL, it defaults to the value of the DISPLAY environment variable.

DESCRIPTION

The **XOpenDisplay** function returns a **Display** structure that serves as the connection to the X server and that contains all the information about that X server. **XOpenDisplay** connects your application to the X server through TCP or DECnet communications protocols, or through some local inter-process communication protocol. If the hostname is a host machine name and a single colon (:) separates the hostname and display number, **XOpenDisplay** connects using TCP streams. If the hostname is not specified, Xlib uses whatever it believes is the fastest transport. If the hostname is a host machine name and a double colon (::) separates the hostname and display number, **XOpenDisplay** connects using DECnet. A single X server can support any or all of these transport mechanisms simultaneously. A particular Xlib implementation can support many more of these transport mechanisms.

If successful, **XOpenDisplay** returns a pointer to a **Display** structure, which is defined in <**X11/Xlib.h**>. If **XOpenDisplay** does not succeed, it returns NULL. After a successful call to **XOpenDisplay**, all of the screens in the display can be used by the client. The screen number specified in the display_name argument is returned by the **DefaultScreen** macro (or the **XDefaultScreen** function). You can access elements of the **Display** and **Screen** structures only by using the information macros or functions. For information about using macros and functions to obtain information from the **Display** structure, see section 2.2.1.

The **XCloseDisplay** function closes the connection to the X server for the display specified in the **Display** structure and destroys all windows, resource IDs (**Window**, **Font**, **Pixmap**, **Colormap**, **Cursor**, and **GContext**), or other resources that the client has created on this display, unless the close-down mode of the resource has been changed (see **XSetCloseDownMode**). Therefore, these windows, resource IDs, and other resources should never be referenced again or an error will be generated. Before exiting, you should call **XCloseDisplay** explicitly so that any pending errors are reported as **XCloseDisplay** performs a final **XSync** operation.

XCloseDisplay can generate a **BadGC** error.

SEE ALSO

AllPlanes(3), XFlush(3), XSetCloseDownMode(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XParseGeometry, XParseColor – parse window geometry and color

SYNTAX

int XParseGeometry(*parsestring*, *x_return*, *y_return*, *width_return*, *height_return*)
char **parsestring*;
int **x_return*, **y_return*;
int **width_return*, **height_return*;

Status XParseColor(*display*, *colormap*, *spec*, *exact_def_return*)
Display **display*;
Colormap *colormap*;
char **spec*;
XColor **exact_def_return*;

ARGUMENTS

colormap Specifies the colormap.

position
default_position Specify the geometry specifications.

display Specifies the connection to the X server.

exact_def_return
Returns the exact color value for later use and sets the **DoRed**, **DoGreen**, and **DoBlue** flags.

fheight
fwidth Specify the font height and width in pixels (increment size).

parsestring Specifies the string you want to parse.

screen Specifies the screen.

spec Specifies the color name string; case is ignored.

width_return
height_return Return the width and height determined.

xadder
yadder Specify additional interior padding needed in the window.

x_return
y_return Return the x and y offsets.

DESCRIPTION

By convention, X applications use a standard string to indicate window size and placement. **XParseGeometry** makes it easier to conform to this standard because it allows you to parse the standard window geometry. Specifically, this function lets you parse strings of the form:

[=][<*width*>x<*height*>][{+-}<*xoffset*>{+-}<*yoffset*>]

The items in this form map into the arguments associated with this function. (Items enclosed in <> are integers, items in [] are optional, and items enclosed in {} indicate "choose one of". Note that the brackets should not appear in the actual string.)

The **XParseGeometry** function returns a bitmask that indicates which of the four values (width, height, xoffset, and yoffset) were actually found in the string and whether the x and y values are negative. By convention, –0 is not equal to +0, because the user needs to be able to say "position the window relative to the right or bottom edge." For each value found, the corresponding argument is updated. For each value not found, the argument is left unchanged. The bits are represented by **XValue**, **YValue**, **WidthValue**, **HeightValue**, **XNegative**, or **YNegative** and are defined in <**X11/Xutil.h**>. They will be set whenever one of the values is defined or one of the signs is set.

If the function returns either the **XValue** or **YValue** flag, you should place the window at the requested position.

The **XParseColor** function provides a simple way to create a standard user interface to color. It takes a string specification of a color, typically from a command line or **XGetDefault** option, and returns the corresponding red, green. and blue values that are suitable for a subsequent call to **XAllocColor** or **XStoreColor**. The color can be specified either as a color name (as in **XAllocNamedColor**) or as an initial sharp sign character followed by a numeric specification, in one of the following formats:

#RGB	(4 bits each)
#RRGGBB	(8 bits each)
#RRRGGGBBB	(12 bits each)
#RRRRGGGGBBBB	(16 bits each)

The R, G, and B represent single hexadecimal digits (both uppercase and lowercase). When fewer than 16 bits each are specified, they represent the most-significant bits of the value. For example, #3a7 is the same as #3000a0007000. The colormap is used only to determine which screen to look up the color on. For example, you can use the screen's default colormap.

If the initial character is a sharp sign but the string otherwise fails to fit the above formats or if the initial character is not a sharp sign and the named color does not exist in the server's database, **XParseColor** fails and returns zero.

XParseColor can generate a **BadColor** error.

DIAGNOSTICS

BadColor A value for a Colormap argument does not name a defined Colormap.

SEE ALSO

XAllocColor(3), XCreateColormap(3), XGetDefault(3), XSetWMProperties(3), XStoreColors(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XPolygonRegion, XClipBox – generate regions

SYNTAX

Region XPolygonRegion(*points*, *n*, *fill_rule*)
XPoint *points*[];
int *n*;
int *fill_rule*;

XClipBox(*r*, *rect_return*)
Region *r*;
XRectangle **rect_return*;

ARGUMENTS

fill_rule	Specifies the fill-rule you want to set for the specified GC. You can pass **EvenOddRule** or **WindingRule**.
n	Specifies the number of points in the polygon.
points	Specifies an array of points.
r	Specifies the region.
rect_return	Returns the smallest enclosing rectangle.

DESCRIPTION

The **XPolygonRegion** function returns a region for the polygon defined by the points array. For an explanation of fill_rule, see **XCreateGC**.

The **XClipBox** function returns the smallest rectangle enclosing the specified region.

SEE ALSO

XCreateGC(3), XDrawPoint(3), XDrawRectangle(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XPropertyEvent – PropertyNotify event structure

STRUCTURES

The structure for **PropertyNotify** events contains:

```
typedef struct {
        int type;                       /* PropertyNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        Atom atom;
        Time time;
        int state;                      /* PropertyNewValue or PropertyDeleted */
} XPropertyEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window whose associated property was changed. The atom member is set to the property's atom and indicates which property was changed or desired. The time member is set to the server time when the property was changed. The state member is set to indicate whether the property was changed to a new value or deleted and can be **PropertyNewValue** or **PropertyDelete**. The state member is set to **PropertyNewValue** when a property of the window is changed using **XChangeProperty** or **XRotateWindowProperties** (even when adding zero-length data using **XChangeProperty**) and when replacing all or part of a property with identical data using **XChangeProperty** or **XRotateWindowProperties**. The state member is set to **PropertyDelete** when a property of the window is deleted using **XDeleteProperty** or, if the delete argument is **True**, **XGetWindowProperty**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGetWindowProperty(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XPutBackEvent – put events back on the queue

SYNTAX

XPutBackEvent(*display, event*)
Display **display*;
XEvent **event*;

ARGUMENTS

display	Specifies the connection to the X server.
event	Specifies a pointer to the event.

DESCRIPTION

The **XPutBackEvent** function pushes an event back onto the head of the display's event queue by copying the event into the queue. This can be useful if you read an event and then decide that you would rather deal with it later. There is no limit to the number of times in succession that you can call **XPutBackEvent**.

SEE ALSO

XAnyEvent(3), XIfEvent(3), XNextEvent(3), XSendEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XPutImage, XGetImage, XGetSubImage – transfer images

SYNTAX

XPutImage(*display, d, gc, image, src_x, src_y, dest_x, dest_y, width, height*)
Display **display*;
Drawable *d*;
GC *gc*;
XImage **image*;
int *src_x, src_y*;
int *dest_x, dest_y*;
unsigned int *width, height*;

XImage *XGetImage(*display, d, x, y, width, height, plane_mask, format*)
Display **display*;
Drawable *d*;
int *x, y*;
unsigned int *width, height*;
unsigned long *plane_mask*;
int *format*;

XImage *XGetSubImage(*display, d, x, y, width, height, plane_mask, format, dest_image, dest_x, dest_y*)
Display **display*;
Drawable *d*;
int *x, y*;
unsigned int *width, height*;
unsigned long *plane_mask*;
int *format*;
XImage **dest_image*;
int *dest_x, dest_y*;

ARGUMENTS

d Specifies the drawable.

dest_image Specify the destination image.

dest_x
dest_y Specify the x and y coordinates, which are relative to the origin of the drawable and are the coordinates of the subimage or which are relative to the origin of the destination rectangle, specify its upper-left corner, and determine where the subimage is placed in the destination image.

display Specifies the connection to the X server.

format Specifies the format for the image. You can pass **XYBitmap**, **XYPixmap**, or **ZPixmap**.

gc Specifies the GC.

image Specifies the image you want combined with the rectangle.

plane_mask Specifies the plane mask.

src_x Specifies the offset in X from the left edge of the image defined by the **XImage** data structure.

src_y Specifies the offset in Y from the top edge of the image defined by the **XImage** data structure.

width
height Specify the width and height of the subimage, which define the dimensions of the

rectangle.

x
y Specify the x and y coordinates, which are relative to the origin of the drawable and define the upper-left corner of the rectangle.

DESCRIPTION

The **XPutImage** function combines an image in memory with a rectangle of the specified drawable. If **XYBitmap** format is used, the depth must be one, or a **BadMatch** error results. The foreground pixel in the GC defines the source for the one bits in the image, and the background pixel defines the source for the zero bits. For **XYPixmap** and **ZPixmap**, the depth must match the depth of the drawable, or a **BadMatch** error results. The section of the image defined by the src_x, src_y, width, and height arguments is drawn on the specified part of the drawable.

This function uses these GC components: function, plane-mask, subwindow-mode, clip-x-origin, clip-y-origin, and clip-mask. It also uses these GC mode-dependent components: foreground and background.

XPutImage can generate **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue** errors.

The **XGetImage** function returns a pointer to an **XImage** structure. This structure provides you with the contents of the specified rectangle of the drawable in the format you specify. If the format argument is **XYPixmap**, the image contains only the bit planes you passed to the plane_mask argument. If the plane_mask argument only requests a subset of the planes of the display, the depth of the returned image will be the number of planes requested. If the format argument is **ZPixmap**, **XGetImage** returns as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits.

XGetImage returns the depth of the image to the depth member of the **XImage** structure. The depth of the image is as specified when the drawable was created, except when getting a subset of the planes in **XYPixmap** format, when the depth is given by the number of bits set to 1 in plane_mask.

If the drawable is a pixmap, the given rectangle must be wholly contained within the pixmap, or a **BadMatch** error results. If the drawable is a window, the window must be viewable, and it must be the case that if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window, or a **BadMatch** error results. Note that the borders of the window can be included and read with this request. If the window has backing-store, the backing-store contents are returned for regions of the window that are obscured by noninferior windows. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window's depth are also undefined. The pointer cursor image is not included in the returned contents. If a problem occurs, **XGetImage** returns NULL.

XGetImage can generate **BadDrawable**, **BadMatch**, and **BadValue** errors.

The **XGetSubImage** function updates dest_image with the specified subimage in the same manner as **XGetImage**. If the format argument is **XYPixmap**, the image contains only the bit planes you passed to the plane_mask argument. If the format argument is **ZPixmap**, **XGetSubImage** returns as zero the bits in all planes not specified in the plane_mask argument. The function performs no range checking on the values in plane_mask and ignores extraneous bits. As a convenience, **XGetSubImage** returns a pointer to the same **XImage** structure specified by dest_image.

The depth of the destination **XImage** structure must be the same as that of the drawable. If the specified subimage does not fit at the specified location on the destination image, the right and bottom edges are clipped. If the drawable is a pixmap, the given rectangle must be wholly

contained within the pixmap, or a **BadMatch** error results. If the drawable is a window, the window must be viewable, and it must be the case that if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen and wholly contained within the outside edges of the window, or a **BadMatch** error results. If the window has backing-store, then the backing-store contents are returned for regions of the window that are obscured by noninferior windows. If the window does not have backing-store, the returned contents of such obscured regions are undefined. The returned contents of visible regions of inferiors of a different depth than the specified window's depth are also undefined. If a problem occurs, **XGetSubImage** returns NULL.

XGetSubImage can generate **BadDrawable**, **BadGC**, **BadMatch**, and **BadValue** errors.

DIAGNOSTICS

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch An **InputOnly** window is used as a Drawable.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XQueryBestSize, XQueryBestTile, XQueryBestStipple – determine efficient sizes

SYNTAX

Status XQueryBestSize(*display, class, which_screen, width, height, width_return, height_return*)
Display **display*;
int *class*;
Drawable *which_screen*;
unsigned int *width, height*;
unsigned int **width_return, *height_return*;

Status XQueryBestTile(*display, which_screen, width, height, width_return, height_return*)
Display **display*;
Drawable *which_screen*;
unsigned int *width, height*;
unsigned int **width_return, *height_return*;

Status XQueryBestStipple(*display, which_screen, width, height, width_return, height_return*)
Display **display*;
Drawable *which_screen*;
unsigned int *width, height*;
unsigned int **width_return, *height_return*;

ARGUMENTS

class	Specifies the class that you are interested in. You can pass **TileShape**, **CursorShape**, or **StippleShape**.
display	Specifies the connection to the X server.
width *height*	Specify the width and height.
which_screen	Specifies any drawable on the screen.
width_return *height_return*	Return the width and height of the object best supported by the display hardware.

DESCRIPTION

The **XQueryBestSize** function returns the best or closest size to the specified size. For **CursorShape**, this is the largest size that can be fully displayed on the screen specified by which_screen. For **TileShape**, this is the size that can be tiled fastest. For **StippleShape**, this is the size that can be stippled fastest. For **CursorShape**, the drawable indicates the desired screen. For **TileShape** and **StippleShape**, the drawable indicates the screen and possibly the window class and depth. An **InputOnly** window cannot be used as the drawable for **TileShape** or **StippleShape**, or a **BadMatch** error results.

XQueryBestSize can generate **BadDrawable**, **BadMatch**, and **BadValue** errors.

The **XQueryBestTile** function returns the best or closest size, that is, the size that can be tiled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. If an **InputOnly** window is used as the drawable, a **BadMatch** error results.

XQueryBestTile can generate **BadDrawable** and **BadMatch** errors.

The **XQueryBestStipple** function returns the best or closest size, that is, the size that can be stippled fastest on the screen specified by which_screen. The drawable indicates the screen and possibly the window class and depth. If an **InputOnly** window is used as the drawable, a **BadMatch** error results.

XQueryBestStipple can generate **BadDrawable** and **BadMatch** errors.

DIAGNOSTICS

BadMatch An **InputOnly** window is used as a Drawable.

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadMatch The values do not exist for an **InputOnly** window.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCreateGC(3), XSetArcMode(3), XSetClipOrigin(3), XSetFillStyle(3), XSetFont(3), XSetLineAttributes(3), XSetState(3), XSetTile(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XQueryColor, XQueryColors, XLookupColor – obtain color values

SYNTAX

XQueryColor(*display, colormap, def_in_out*)
Display **display*;
Colormap *colormap*;
XColor **def_in_out*;

XQueryColors(*display, colormap, defs_in_out, ncolors*)
Display **display*;
Colormap *colormap*;
XColor *defs_in_out*[];
int *ncolors*;

Status XLookupColor(*display, colormap, color_name, exact_def_return, screen_def_return*)
Display **display*;
Colormap *colormap*;
char **color_name*;
XColor **exact_def_return*, **screen_def_return*;

ARGUMENTS

colormap Specifies the colormap.

color_name Specifies the color name string (for example, red) whose color definition structure you want returned.

def_in_out Specifies and returns the RGB values for the pixel specified in the structure.

defs_in_out Specifies and returns an array of color definition structures for the pixel specified in the structure.

display Specifies the connection to the X server.

exact_def_return
Returns the exact RGB values.

ncolors Specifies the number of **XColor** structures in the color definition array.

screen_def_return
Returns the closest RGB values provided by the hardware.

DESCRIPTION

The **XQueryColor** function returns the hardware-specific RGB values for each pixel in the **XColor** structures and sets the **DoRed**, **DoGreen**, and **DoBlue** flags. The **XQueryColors** function returns the RGB values for each pixel in the **XColor** structures and sets the **DoRed**, **DoGreen**, and **DoBlue** flags.

XQueryColor and **XQueryColors** can generate **BadColor** and **BadValue** errors.

The **XLookupColor** function looks up the string name of a color with respect to the screen associated with the specified colormap. It returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified colormap. You should use the ISO Latin-1 encoding; uppercase and lowercase do not matter. **XLookupColor** returns nonzero if the name existed in the color database or zero if it did not exist.

DIAGNOSTICS

BadColor A value for a Colormap argument does not name a defined Colormap.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XAllocColor(3), XCreateColormap(3), XStoreColors(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XQueryPointer – get pointer coordinates

SYNTAX

Bool XQueryPointer(*display, w, root_return, child_return, root_x_return, root_y_return, win_x_return, win_y_return, mask_return*)

Display **display*;
Window *w*;
Window **root_return*, **child_return*;
int **root_x_return*, **root_y_return*;
int **win_x_return*, **win_y_return*;
unsigned int **mask_return*;

ARGUMENTS

child_return	Returns the child window that the pointer is located in, if any.
display	Specifies the connection to the X server.
mask_return	Returns the current state of the modifier keys and pointer buttons.
root_return	Returns the root window that the pointer is in.
root_x_return *root_y_return*	Return the pointer coordinates relative to the root window's origin.
w	Specifies the window.
win_x_return *win_y_return*	Return the pointer coordinates relative to the specified window.

DESCRIPTION

The **XQueryPointer** function returns the root window the pointer is logically on and the pointer coordinates relative to the root window's origin. If **XQueryPointer** returns **False**, the pointer is not on the same screen as the specified window, and **XQueryPointer** returns **None** to child_return and zero to win_x_return and win_y_return. If **XQueryPointer** returns **True**, the pointer coordinates returned to win_x_return and win_y_return are relative to the origin of the specified window. In this case, **XQueryPointer** returns the child that contains the pointer, if any, or else **None** to child_return.

XQueryPointer returns the current logical state of the keyboard buttons and the modifier keys in mask_return. It sets mask_return to the bitwise inclusive OR of one or more of the button or modifier key bitmasks to match the current state of the mouse buttons and the modifier keys.

XQueryPointer can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XGetWindowAttributes(3), XQueryTree(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XQueryTree – query window tree information

SYNTAX

Status XQueryTree(*display, w, root_return, parent_return, children_return, nchildren_return*)
Display **display*;
Window *w*;
Window **root_return*;
Window **parent_return*;
Window ***children_return*;
unsigned int **nchildren_return*;

ARGUMENTS

children_return Returns a pointer to the list of children.

display Specifies the connection to the X server.

nchildren_return
Returns the number of children.

parent_return Returns the parent window.

root_return Returns the root window.

w Specifies the window whose list of children, root, parent, and number of children you want to obtain.

DESCRIPTION

The **XQueryTree** function returns the root ID, the parent window ID, a pointer to the list of children windows, and the number of children in the list for the specified window. The children are listed in current stacking order, from bottommost (first) to topmost (last). **XQueryTree** returns zero if it fails and nonzero if it succeeds. To free this list when it is no longer needed, use **XFree**.

BUGS

This really should return a screen *, not a root window ID.

SEE ALSO

XFree(3), XGetWindowAttributes(3), XQueryPointer(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XRaiseWindow, XLowerWindow, XCirculateSubwindows, XCirculateSubwindowsUp, XCirculateSubwindowsDown, XRestackWindows – change window stacking order

SYNTAX

XRaiseWindow(*display, w*)
Display **display*;
Window *w*;

XLowerWindow(*display, w*)
Display **display*;
Window *w*;

XCirculateSubwindows(*display, w, direction*)
Display **display*;
Window *w*;
int *direction*;

XCirculateSubwindowsUp(*display, w*)
Display **display*;
Window *w*;

XCirculateSubwindowsDown(*display, w*)
Display **display*;
Window *w*;

XRestackWindows(*display, windows, nwindows*);
Display **display*;
Window *windows*[];
int *nwindows*;

ARGUMENTS

direction	Specifies the direction (up or down) that you want to circulate the window. You can pass **RaiseLowest** or **LowerHighest**.
display	Specifies the connection to the X server.
nwindows	Specifies the number of windows to be restacked.
w	Specifies the window.
windows	Specifies an array containing the windows to be restacked.

DESCRIPTION

The **XRaiseWindow** function raises the specified window to the top of the stack so that no sibling window obscures it. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack but leaving its x and y location on the desk constant. Raising a mapped window may generate **Expose** events for the window and any mapped subwindows that were formerly obscured.

If the override-redirect attribute of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no processing is performed. Otherwise, the window is raised.

XRaiseWindow can generate a **BadWindow** error.

The **XLowerWindow** function lowers the specified window to the bottom of the stack so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack but leaving its x and y location on the desk constant. Lowering a mapped window will generate **Expose** events on any windows it formerly obscured.

If the override-redirect attribute of the window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates a **ConfigureRequest** event, and no processing is performed. Otherwise, the window is lowered to the bottom of the stack.

XLowerWindow can generate a **BadWindow** error.

The **XCirculateSubwindows** function circulates children of the specified window in the specified direction. If you specify **RaiseLowest**, **XCirculateSubwindows** raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. If you specify **LowerHighest**, **XCirculateSubwindows** lowers the highest mapped child (if any) that occludes another child to the bottom of the stack. Exposure processing is then performed on formerly obscured windows. If some other client has selected **SubstructureRedirectMask** on the window, the X server generates a **CirculateRequest** event, and no further processing is performed. If a child is actually restacked, the X server generates a **CirculateNotify** event.

XCirculateSubwindows can generate **BadValue** and **BadWindow** errors.

The **XCirculateSubwindowsUp** function raises the lowest mapped child of the specified window that is partially or completely occluded by another child. Completely unobscured children are not affected. This is a convenience function equivalent to **XCirculateSubwindows** with **RaiseLowest** specified.

XCirculateSubwindowsUp can generate a **BadWindow** error.

The **XCirculateSubwindowsDown** function lowers the highest mapped child of the specified window that partially or completely occludes another child. Completely unobscured children are not affected. This is a convenience function equivalent to **XCirculateSubwindows** with **LowerHighest** specified.

XCirculateSubwindowsDown can generate a **BadWindow** error.

The **XRestackWindows** function restacks the windows in the order specified, from top to bottom. The stacking order of the first window in the windows array is unaffected, but the other windows in the array are stacked underneath the first window, in the order of the array. The stacking order of the other windows is not affected. For each window in the window array that is not a child of the specified window, a **BadMatch** error results.

If the override-redirect attribute of a window is **False** and some other client has selected **SubstructureRedirectMask** on the parent, the X server generates **ConfigureRequest** events for each window whose override-redirect flag is not set, and no further processing is performed. Otherwise, the windows will be restacked in top to bottom order.

XRestackWindows can generate **BadWindow** error.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XConfigureWindow(3), XCreateWindow(3), XDestroyWindow(3), XMapWindow(3), XUnmapWindow(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XReadBitmapFile, XWriteBitmapFile, XCreatePixmapFromBitmapData, XCreateBitmapFromData – manipulate bitmaps

SYNTAX

int XReadBitmapFile(*display, d, filename, width_return, height_return, bitmap_return, x_hot_return,*
y_hot_return)
Display **display*;
Drawable *d*;
char **filename*;
unsigned int **width_return*, **height_return*;
Pixmap **bitmap_return*;
int **x_hot_return*, **y_hot_return*;

int XWriteBitmapFile(*display, filename, bitmap, width, height, x_hot, y_hot*)
Display **display*;
char **filename*;
Pixmap *bitmap*;
unsigned int *width, height*;
int *x_hot, y_hot*;

Pixmap XCreatePixmapFromBitmapData(*display, d, data, width, height, fg, bg, depth*)
Display **display*;
Drawable *d*;
char **data*;
unsigned int *width, height*;
unsigned long *fg, bg*;
unsigned int *depth*;

Pixmap XCreateBitmapFromData(*display, d, data, width, height*)
Display **display*;
Drawable *d*;
char **data*;
unsigned int *width, height*;

ARGUMENTS

bitmap	Specifies the bitmap.
bitmap_return	Returns the bitmap that is created.
d	Specifies the drawable that indicates the screen.
data	Specifies the data in bitmap format.
data	Specifies the location of the bitmap data.
depth	Specifies the depth of the pixmap.
display	Specifies the connection to the X server.
fg *bg*	Specify the foreground and background pixel values to use.
filename	Specifies the file name to use. The format of the file name is operating-system dependent.
width *height*	Specify the width and height.
width_return *height_return*	Return the width and height values of the read in bitmap file.

x_hot
y_hot Specify where to place the hotspot coordinates (or –1,–1 if none are present) in the file.

x_hot_return
y_hot_return Return the hotspot coordinates.

DESCRIPTION

The **XReadBitmapFile** function reads in a file containing a bitmap. The ability to read other than the standard format is implementation dependent. If the file cannot be opened, **XReadBitmapFile** returns **BitmapOpenFailed**. If the file can be opened but does not contain valid bitmap data, it returns **BitmapFileInvalid**. If insufficient working storage is allocated, it returns **BitmapNoMemory**. If the file is readable and valid, it returns **BitmapSuccess**.

XReadBitmapFile returns the bitmap's height and width, as read from the file, to width_return and height_return. It then creates a pixmap of the appropriate size, reads the bitmap data from the file into the pixmap, and assigns the pixmap to the caller's variable bitmap. The caller must free the bitmap using **XFreePixmap** when finished. If *name*_x_hot and *name*_y_hot exist, **XReadBitmapFile** returns them to x_hot_return and y_hot_return; otherwise, it returns –1,–1.

XReadBitmapFile can generate **BadAlloc** and **BadDrawable** errors.

The **XWriteBitmapFile** function writes a bitmap out to a file in the X version 11 format. If the file cannot be opened for writing, it returns **BitmapOpenFailed**. If insufficient memory is allocated, **XWriteBitmapFile** returns **BitmapNoMemory**; otherwise, on no error, it returns **BitmapSuccess**. If x_hot and y_hot are not –1, –1, **XWriteBitmapFile** writes them out as the hotspot coordinates for the bitmap.

XWriteBitmapFile can generate **BadDrawable** and **BadMatch** errors.

The **XCreatePixmapFromBitmapData** function creates a pixmap of the given depth and then does a bitmap-format **XPutImage** of the data into it. The depth must be supported by the screen of the specified drawable, or a **BadMatch** error results.

XCreatePixmapFromBitmapData can generate **BadAlloc** and **BadMatch** errors.

The **XCreateBitmapFromData** function allows you to include in your C program (using #include) a bitmap file that was written out by **XWriteBitmapFile** (X version 11 format only) without reading in the bitmap file. The following example creates a gray bitmap:

```
#include "gray.bitmap"

Pixmap bitmap;
bitmap = XCreateBitmapFromData(display, window, gray_bits, gray_width, gray_height);
```

If insufficient working storage was allocated, **XCreateBitmapFromData** returns **None**. It is your responsibility to free the bitmap using **XFreePixmap** when finished.

XCreateBitmapFromData can generate a **BadAlloc** error.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

BadMatch An **InputOnly** window is used as a Drawable.

SEE ALSO

XCreatePixmap(3), XPutImage(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XRecolorCursor, XFreeCursor, XQueryBestCursor – manipulate cursors

SYNTAX

XRecolorCursor(*display, cursor, foreground_color, background_color*)
Display **display*;
Cursor *cursor*;
XColor **foreground_color*, **background_color*;

XFreeCursor(*display, cursor*)
Display **display*;
Cursor *cursor*;

Status XQueryBestCursor(*display, d, width, height, width_return, height_return*)
Display **display*;
Drawable *d*;
unsigned int *width, height*;
unsigned int **width_return*, **height_return*;

ARGUMENTS

background_color
Specifies the RGB values for the background of the source.

cursor Specifies the cursor.

d Specifies the drawable, which indicates the screen.

display Specifies the connection to the X server.

foreground_color
Specifies the RGB values for the foreground of the source.

width
height Specify the width and heightof the cursor that you want the size information for.

width_return
height_return Return the best width and height that is closest to the specified width and height.

DESCRIPTION

The **XRecolorCursor** function changes the color of the specified cursor, and if the cursor is being displayed on a screen, the change is visible immediately.

XRecolorCursor can generate a **BadCursor** error.

The **XFreeCursor** function deletes the association between the cursor resource ID and the specified cursor. The cursor storage is freed when no other resource references it. The specified cursor ID should not be referred to again.

XFreeCursor can generate a **BadCursor** error.

Some displays allow larger cursors than other displays. The **XQueryBestCursor** function provides a way to find out what size cursors are actually possible on the display. It returns the largest size that can be displayed. Applications should be prepared to use smaller cursors on displays that cannot support large ones.

XQueryBestCursor can generate a **BadDrawable** error.

DIAGNOSTICS

BadCursor A value for a Cursor argument does not name a defined Cursor.

BadDrawable A value for a Drawable argument does not name a defined Window or Pixmap.

SEE ALSO

XCreateColormap(3), XCreateFontCursor(3), XDefineCusor(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XReparentEvent – ReparentNotify event structure

STRUCTURES

The structure for **ReparentNotify** events contains:

```
typedef struct {
        int type;                       /* ReparentNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
        Window parent;
        int x, y;
        Bool override_redirect;
} XReparentEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the reparented window or to the old or the new parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. The window member is set to the window that was reparented. The parent member is set to the new parent window. The x and y members are set to the reparented window's coordinates relative to the new parent window's origin and define the upper-left outer corner of the reparented window. The override_redirect member is set to the override-redirect attribute of the window specified by the window member. Window manager clients normally should ignore this window if the override_redirect member is **True**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XReparentWindow – reparent windows

SYNTAX

XReparentWindow(*display, w, parent, x, y*)
Display **display*;
Window *w*;
Window *parent*;
int *x*, *y*;

ARGUMENTS

display	Specifies the connection to the X server.
parent	Specifies the parent window.
w	Specifies the window.
x *y*	Specify the x and y coordinatesof the position in the new parent window.

DESCRIPTION

If the specified window is mapped, **XReparentWindow** automatically performs an **UnmapWindow** request on it, removes it from its current position in the hierarchy, and inserts it as the child of the specified parent. The window is placed in the stacking order on top with respect to sibling windows.

After reparenting the specified window, **XReparentWindow** causes the X server to generate a **ReparentNotify** event. The override_redirect member returned in this event is set to the window's corresponding attribute. Window manager clients usually should ignore this window if this member is set to **True**. Finally, if the specified window was originally mapped, the X server automatically performs a **MapWindow** request on it.

The X server performs normal exposure processing on formerly obscured windows. The X server might not generate **Expose** events for regions from the initial **UnmapWindow** request that are immediately obscured by the final **MapWindow** request. A **BadMatch** error results if:

- The new parent window is not on the same screen as the old parent window.
- The new parent window is the specified window or an inferior of the specified window.
- The specified window has a **ParentRelative** background, and the new parent window is not the same depth as the specified window.

XReparentWindow can generate **BadMatch** and **BadWindow** errors.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeSaveSet(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XResizeRequestEvent – ResizeRequest event structure

STRUCTURES

The structure for **ResizeRequest** events contains:

```
typedef struct {
        int type;                       /* ResizeRequest */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        int width, height;
} XResizeRequestEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window whose size another client attempted to change. The width and height members are set to the inside size of the window, excluding the border.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSaveContext, XFindContext, XDeleteContext, XUniqueContext – associative look-up routines

SYNTAX

int XSaveContext(*display, w, context, data*)
Display **display*;
Window *w*;
XContext *context*;
caddr_t *data*;

int XFindContext(*display, w, context, data_return*)
Display **display*;
Window *w*;
XContext *context*;
caddr_t **data_return*;

int XDeleteContext(*display, w, context*)
Display **display*;
Window *w*;
XContext *context*;

XContext XUniqueContext()

ARGUMENTS

context	Specifies the context type to which the data belongs.
data	Specifies the data to be associated with the window and type.
data_return	Returns a pointer to the data.
display	Specifies the connection to the X server.
w	Specifies the window with which the data is associated.

DESCRIPTION

If an entry with the specified window and type already exists, **XSaveContext** overrides it with the specified context. The **XSaveContext** function returns a nonzero error code if an error has occurred and zero otherwise. Possible errors are **XCNOMEM** (out of memory).

Because it is a return value, the data is a pointer. The **XFindContext** function returns a nonzero error code if an error has occurred and zero otherwise. Possible errors are **XCNOENT** (context-not-found).

The **XDeleteContext** function deletes the entry for the given window and type from the data structure. This function returns the same error codes that **XFindContext** returns if called with the same arguments. **XDeleteContext** does not free the data whose address was saved.

The **XUniqueContext** function creates a unique context type that may be used in subsequent calls to **XSaveContext**.

SEE ALSO

Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSelectInput – select input events

SYNTAX

XSelectInput(*display, w, event_mask*)
Display **display*;
Window *w*;
long *event_mask*;

ARGUMENTS

display	Specifies the connection to the X server.
event_mask	Specifies the event mask.
w	Specifies the window whose events you are interested in.

DESCRIPTION

The **XSelectInput** function requests that the X server report the events associated with the specified event mask. Initially, X will not report any of these events. Events are reported relative to a window. If a window is not interested in a device event, it usually propagates to the closest ancestor that is interested, unless the do_not_propagate mask prohibits it.

Setting the event-mask attribute of a window overrides any previous call for the same window but not for other clients. Multiple clients can select for the same events on the same window with the following restrictions:

- Multiple clients can select events on the same window because their event masks are disjoint. When the X server generates an event, it reports it to all interested clients.
- Only one client at a time can select **CirculateRequest**, **ConfigureRequest**, or **MapRequest** events, which are associated with the event mask **SubstructureRedirectMask**.
- Only one client at a time can select a **ResizeRequest** event, which is associated with the event mask **ResizeRedirectMask**.
- Only one client at a time can select a **ButtonPress** event, which is associated with the event mask **ButtonPressMask**.

The server reports the event to all interested clients.

XSelectInput can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSelectionClearEvent – SelectionClear event structure

STRUCTURES

The structure for **SelectionClear** events contains:

```
typedef struct {
        int type;                       /* SelectionClear */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        Atom selection;
        Time time;
} XSelectionClearEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window losing ownership of the selection. The selection member is set to the selection atom. The time member is set to the last change time recorded for the selection. The owner member is the window that was specified by the current owner in its **XSetSelectionOwner** call.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XSetSelectionOwner(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSelectionEvent – SelectionNotify event structure

STRUCTURES

The structure for **SelectionNotify** events contains:

```
typedef struct {
        int type;                       /* SelectionNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window requestor;
        Atom selection;
        Atom target;
        Atom property;                  /* atom or None */
        Time time;
} XSelectionEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The requestor member is set to the window associated with the requestor of the selection. The selection member is set to the atom that indicates the selection. For example, PRIMARY is used for the primary selection. The target member is set to the atom that indicates the converted type. For example, PIXMAP is used for a pixmap. The property member is set to the atom that indicates which property the result was stored on. If the conversion failed, the property member is set to **None**. The time member is set to the time the conversion took place and can be a timestamp or **CurrentTime**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSelectionRequestEvent – SelectionRequest event structure

STRUCTURES

The structure for **SelectionRequest** events contains:

```
typedef struct {
        int type;                       /* SelectionRequest */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window owner;
        Window requestor;
        Atom selection;
        Atom target;
        Atom property;
        Time time;
} XSelectionRequestEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The owner member is set to the window owning the selection and is the window that was specified by the current owner in its **XSetSelectionOwner** call. The requestor member is set to the window requesting the selection. The selection member is set to the atom that names the selection. For example, PRIMARY is used to indicate the primary selection. The target member is set to the atom that indicates the type the selection is desired in. The property member can be a property name or **None**. The time member is set to the time and is a timestamp or **CurrentTime** from the **ConvertSelection** request.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSetSelectionOwner(3), XUnmapEvent(3), XVisibilityEvent(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSendEvent, XDisplayMotionBufferSize, XGetMotionEvents, XTimeCoord – send events and pointer motion history structure

SYNTAX

Status XSendEvent(*display, w, propagate, event_mask, event_send*)
Display **display*;
Window *w*;
Bool *propagate*;
long *event_mask*;
XEvent **event_send*;

unsigned long XDisplayMotionBufferSize(*display*)
Display **display*;

XTimeCoord *XGetMotionEvents(*display, w, start, stop, nevents_return*)
Display **display*;
Window *w*;
Time *start, stop*;
int **nevents_return*;

ARGUMENTS

display	Specifies the connection to the X server.
event_mask	Specifies the event mask.
event_send	Specifies a pointer to the event that is to be sent.
nevents_return	Returns the number of events from the motion history buffer.
propagate	Specifies a Boolean value.
start	
stop	Specify the time interval in which the events are returned from the motion history buffer. You can pass a timestamp or **CurrentTime**. **PointerWindow**,
w	Specifies the window the window the event is to be sent to,.

DESCRIPTION

The **XSendEvent** function identifies the destination window, determines which clients should receive the specified events, and ignores any active grabs. This function requires you to pass an event mask. For a discussion of the valid event mask names, see section 8.3. This function uses the w argument to identify the destination window as follows:

- If w is **PointerWindow**, the destination window is the window that contains the pointer.
- If w is **InputFocus** and if the focus window contains the pointer, the destination window is the window that contains the pointer; otherwise, the destination window is the focus window.

To determine which clients should receive the specified events, **XSendEvent** uses the propagate argument as follows:

- If event_mask is the empty set, the event is sent to the client that created the destination window. If that client no longer exists, no event is sent.
- If propagate is **False**, the event is sent to every client selecting on destination any of the event types in the event_mask argument.
- If propagate is **True** and no clients have selected on destination any of the event types in event-mask, the destination is replaced with the closest ancestor of destination for which some client has selected a type in event-mask and for which no intervening window has that type in its do-not-propagate-mask. If no such window exists or if the window is an ancestor of the focus window and **InputFocus** was originally specified as the destination, the event

is not sent to any clients. Otherwise, the event is reported to every client selecting on the final destination any of the types specified in event_mask.

The event in the **XEvent** structure must be one of the core events or one of the events defined by an extension (or a **BadValue** error results) so that the X server can correctly byte-swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the X server except to force send_event to **True** in the forwarded event and to set the serial number in the event correctly.

XSendEvent returns zero if the conversion to wire protocol format failed and returns nonzero otherwise. **XSendEvent** can generate **BadValue** and **BadWindow** errors.

The server may retain the recent history of the pointer motion and do so to a finer granularity than is reported by **MotionNotify** events. The **XGetMotionEvents** function makes this history available.

The **XGetMotionEvents** function returns all events in the motion history buffer that fall between the specified start and stop times, inclusive, and that have coordinates that lie within the specified window (including its borders) at its present placement. If the start time is later than the stop time or if the start time is in the future, no events are returned. If the stop time is in the future, it is equivalent to specifying **CurrentTime**. **XGetMotionEvents** can generate a **BadWindow**e error.

STRUCTURES

The **XTimeCoord** structure contains:

```
typedef struct {
        Time time;
        short x, y;
} XTimeCoord;
```

The time member is set to the time, in milliseconds. The x and y members are set to the coordinates of the pointer and are reported relative to the origin of the specified window.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAnyEvent(3), XIfEvent(3), XNextEvent(3), XPutBackEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetArcMode, XSetSubwindowMode, XSetGraphicsExposure – GC convenience routines

SYNTAX

XSetArcMode(*display, gc, arc_mode*)
Display **display*;
GC *gc*;
int *arc_mode*;

XSetSubwindowMode(*display, gc, subwindow_mode*)
Display **display*;
GC *gc*;
int *subwindow_mode*;

XSetGraphicsExposures(*display, gc, graphics_exposures*)
Display **display*;
GC *gc*;
Bool *graphics_exposures*;

ARGUMENTS

arc_mode Specifies the arc mode. You can pass **ArcChord** or **ArcPieSlice**.

display Specifies the connection to the X server.

gc Specifies the GC.

graphics_exposures
Specifies a Boolean value that indicates whether you want **GraphicsExpose** and **NoExpose** events to be reported when calling **XCopyArea** and **XCopyPlane** with this GC.

subwindow_mode
Specifies the subwindow mode. You can pass **ClipByChildren** or **IncludeInferiors**.

DESCRIPTION

The **XSetArcMode** function sets the arc mode in the specified GC.

XSetArcMode can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

The **XSetSubwindowMode** function sets the subwindow mode in the specified GC.

XSetSubwindowMode can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

The **XSetGraphicsExposures** function sets the graphics-exposures flag in the specified GC.

XSetGraphicsExposures can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadGC A value for a GContext argument does not name a defined GContext.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCopyArea(3), XCreateGC(3), XQueryBestSize(3), XSetClipOrigin(3), XSetFillStyle(3), XSetFont(3), XSetLineAttributes(3), XSetState(3), XSetTile(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetClipOrigin, XSetClipMask, XSetClipRectangles – GC convenience routines

SYNTAX

XSetClipOrigin(*display, gc, clip_x_origin, clip_y_origin*)
Display **display*;
GC *gc*;
int *clip_x_origin, clip_y_origin*;

XSetClipMask(*display, gc, pixmap*)
Display **display*;
GC *gc*;
Pixmap *pixmap*;

XSetClipRectangles(*display, gc, clip_x_origin, clip_y_origin, rectangles, n, ordering*)
Display **display*;
GC *gc*;
int *clip_x_origin, clip_y_origin*;
XRectangle *rectangles*[];
int *n*;
int *ordering*;

ARGUMENTS

display	Specifies the connection to the X server.
clip_x_origin *clip_y_origin*	Specify the x and y coordinates of the clip-mask origin.
gc	Specifies the GC.
n	Specifies the number of rectangles.
ordering	Specifies the ordering relations on the rectangles. You can pass **Unsorted**, **YSorted**, **YXSorted**, or **YXBanded**.
pixmap	Specifies the pixmap or **None**.
rectangles	Specifies an array of rectangles that define the clip-mask.

DESCRIPTION

The **XSetClipOrigin** function sets the clip origin in the specified GC. The clip-mask origin is interpreted relative to the origin of whatever destination drawable is specified in the graphics request.

XSetClipOrigin can generate **BadAlloc** and **BadGC** errors.

The **XSetClipMask** function sets the clip-mask in the specified GC to the specified pixmap. If the clip-mask is set to **None**, the pixels are are always drawn (regardless of the clip-origin).

XSetClipMask can generate **BadAlloc**, **BadGC**, **BadMatch**, and **BadValue** errors.

The **XSetClipRectangles** function changes the clip-mask in the specified GC to the specified list of rectangles and sets the clip origin. The output is clipped to remain contained within the rectangles. The clip-origin is interpreted relative to the origin of whatever destination drawable is specified in a graphics request. The rectangle coordinates are interpreted relative to the clip-origin. The rectangles should be nonintersecting, or the graphics results will be undefined. Note that the list of rectangles can be empty, which effectively disables output. This is the opposite of passing **None** as the clip-mask in **XCreateGC**, **XChangeGC**, and **XSetClipMask**.

If known by the client, ordering relations on the rectangles can be specified with the ordering argument. This may provide faster operation by the server. If an incorrect ordering is specified, the X server may generate a **BadMatch** error, but it is not required to do so. If no error is generated, the graphics results are undefined. **Unsorted** means the rectangles are in arbitrary order.

YSorted means that the rectangles are nondecreasing in their Y origin. **YXSorted** additionally constrains **YSorted** order in that all rectangles with an equal Y origin are nondecreasing in their X origin. **YXBanded** additionally constrains **YXSorted** by requiring that, for every possible Y scanline, all rectangles that include that scanline have an identical Y origins and Y extents.

XSetClipRectangles can generate **BadAlloc**, **BadGC**, **BadMatch**, and **BadValue** errors.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadGC A value for a GContext argument does not name a defined GContext.

BadMatch Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCreateGC(3), XDrawRectangle(3), XQueryBestSize(3), XSetArcMode(3), XSetFillStyle(3), XSetFont(3), XSetLineAttributes(3), XSetState(3), XSetTile(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetCloseDownMode, XKillClient – control clients

SYNTAX

XSetCloseDownMode(*display, close_mode*)
Display **display*;
int *close_mode*;

XKillClient(*display, resource*)
Display **display*;
XID *resource*;

ARGUMENTS

close_mode Specifies the client close-down mode. You can pass **DestroyAll**, **RetainPermanent**, or **RetainTemporary**.

display Specifies the connection to the X server.

resource Specifies any resource associated with the client that you want to destroy or **AllTemporary**.

DESCRIPTION

The **XSetCloseDownMode** defines what will happen to the client's resources at connection close. A connection starts in **DestroyAll** mode. For information on what happens to the client's resources when the close_mode argument is **RetainPermanent** or **RetainTemporary**, see section 2.6.

XSetCloseDownMode can generate a **BadValue** error.

The **XKillClient** function forces a close-down of the client that created the resource if a valid resource is specified. If the client has already terminated in either **RetainPermanent** or **RetainTemporary** mode, all of the client's resources are destroyed. If **AllTemporary** is specified, the resources of all clients that have terminated in **RetainTemporary** are destroyed (see section 2.6). This permits implementation of window manager facilities that aid debugging. A client can set its close-down mode to **RetainTemporary**. If the client then crashes, its windows would not be destroyed. The programmer can then inspect the application's window tree and use the window manager to destroy the zombie windows.

XKillClient can generate a **BadValue** error.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetCommand, XGetCommand – set or read a window's WM_COMMAND property

SYNTAX

XSetCommand(*display, w, argv, argc*)
Display **display*;
Window *w*;
char ***argv*;
int *argc*;

Status XGetCommand(*display, w, argv_return, argc_return*)
Display **display*;
Window *w*;
char ****argv_return*;
int **argc_return*;

ARGUMENTS

argc	Specifies the number of arguments.
argc_return	Returns the number of arguments returned.
argv	Specifies the application's argument list.
argv_return	Returns the application's argument list.
display	Specifies the connection to the X server.
w	Specifies the window.

DESCRIPTION

The **XSetCommand** function sets the command and arguments used to invoke the application. (Typically, argv is the argv array of your main program.)

XSetCommand can generate **BadAlloc** and **BadWindow** errors.

The **XGetCommand** function reads the WM_COMMAND property from the specified window and returns a string list. If the WM_COMMAND property exists, it is of type STRING and format 8. If sufficient memory can be allocated to contain the string list, **XGetCommand** fills in the argv_return and argc_return arguments and returns a non-zero status. Otherwise, it returns a zero status. To free the memory allocated to the string list, use **XFreeStringList**.

PROPERTIES

WM_COMMAND
The command and arguments, separated by ASCII nulls, used to invoke the application.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadWindow	A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetErrorHandler, XGetErrorText, XDisplayName, XSetIOErrorHandler, XGetErrorDatabaseText - default error handlers

SYNTAX

int (*XSetErrorHandler(*handler*))()
    int (**handler*)(Display *, XErrorEvent *)

XGetErrorText(*display, code, buffer_return, length*)
    Display **display*;
    int *code*;
    char **buffer_return*;
    int *length*;

char *XDisplayName(*string*)
    char **string*;

int (*XSetIOErrorHandler(*handler*))()
    int (**handler*)(Display *);

XGetErrorDatabaseText(*display, name, message, default_string, buffer_return, length*)
    Display **display*;
    char **name*, **message*;
    char **default_string*;
    char **buffer_return*;
    int *length*;

ARGUMENTS

buffer_return	Returns the error description.
code	Specifies the error code for which you want to obtain a description.
default_string	Specifies the default error message if none is found in the database.
display	Specifies the connection to the X server.
handler	Specifies the program's supplied error handler.
length	Specifies the size of the buffer.
message	Specifies the type of the error message.
name	Specifies the name of the application.
string	Specifies the character string.

DESCRIPTION

Xlib generally calls the program's supplied error handler whenever an error is received. It is not called on **BadName** errors from **OpenFont**, **LookupColor**, or **AllocNamedColor** protocol requests or on **BadFont** errors from a **QueryFont** protocol request. These errors generally are reflected back to the program through the procedural interface. Because this condition is not assumed to be fatal, it is acceptable for your error handler to return. However, the error handler should not call any functions (directly or indirectly) on the display that will generate protocol requests or that will look for input events. The previous error handler is returned.

The **XGetErrorText** function copies a null-terminated string describing the specified error code into the specified buffer. It is recommended that you use this function to obtain an error description because extensions to Xlib may define their own error codes and error strings.

The **XDisplayName** function returns the name of the display that **XOpenDisplay** would attempt to use. If a NULL string is specified, **XDisplayName** looks in the environment for the display and returns the display name that **XOpenDisplay** would attempt to use. This makes it easier to report to the user precisely which display the program attempted to open when the initial connection attempt failed.

The **XSetIOErrorHandler** sets the fatal I/O error handler. Xlib calls the program's supplied error handler if any sort of system call error occurs (for example, the connection to the server was lost). This is assumed to be a fatal condition, and the called routine should not return. If the I/O error handler does return, the client process exits.

Note that the previous error handler is returned.

The **XGetErrorDatabaseText** function returns a message (or the default message) from the error message database. Xlib uses this function internally to look up its error messages. On a UNIX-based system, the error message database is **/usr/lib/X11/XErrorDB**.

The name argument should generally be the name of your application. The message argument should indicate which type of error message you want. Xlib uses three predefined message types to report errors (uppercase and lowercase matter):

XProtoError	The protocol error number is used as a string for the message argument.
XlibMessage	These are the message strings that are used internally by the library.
XRequest	For a core protocol request, the major request protocol number is used for the message argument. For an extension request, the extension name (as given by **InitExtension**) followed by a period (.) and the minor request protocol number is used for the message argument. If no string is found in the error database, the default_string is returned to the buffer argument.

SEE ALSO

XOpenDisplay(3), XSynchronize(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetFillStyle, XSetFillRule – GC convenience routines

SYNTAX

XSetFillStyle(*display, gc, fill_style*)
Display **display*;
GC *gc*;
int *fill_style*;

XSetFillRule(*display, gc, fill_rule*)
Display **display*;
GC *gc*;
int *fill_rule*;

ARGUMENTS

display	Specifies the connection to the X server.
fill_rule	Specifies the fill-rule you want to set for the specified GC. You can pass **EvenOddRule** or **WindingRule**.
fill_style	Specifies the fill-style you want to set for the specified GC. You can pass **FillSolid**, **FillTiled**, **FillStippled**, or **FillOpaqueStippled**.
gc	Specifies the GC.

DESCRIPTION

The **XSetFillStyle** function sets the fill-style in the specified GC.

XSetFillStyle can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

The **XSetFillRule** function sets the fill-rule in the specified GC.

XSetFillRule can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadGC	A value for a GContext argument does not name a defined GContext.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCreateGC(3), XQueryBestSize(3), XSetArcMode(3), XSetClipOrigin(3), XSetFont(3), XSetLineAttributes(3), XSetState(3), XSetTile(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetFont - GC convenience routines

SYNTAX

XSetFont(*display*, *gc*, *font*)
Display **display*;
GC *gc*;
Font *font*;

ARGUMENTS

display	Specifies the connection to the X server.
font	Specifies the font.
gc	Specifies the GC.

DESCRIPTION

The **XSetFont** function sets the current font in the specified GC.

XSetFont can generate **BadAlloc**, **BadFont**, and **BadGC** errors.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadFont	A value for a Font or GContext argument does not name a defined Font.
BadGC	A value for a GContext argument does not name a defined GContext.

SEE ALSO

XCreateGC(3), XQueryBestSize(3), XSetArcMode(3), XSetClipOrigin(3), XSetFillStyle(3), XSetLineAttributes(3), XSetState(3), XSetTile(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetFontPath, XGetFontPath, XFreeFontPath – set, get, or free the font search path

SYNTAX

XSetFontPath(*display, directories, ndirs*)
Display **display*;
char ***directories*;
int *ndirs*;

char **XGetFontPath(*display, npaths_return*)
Display **display*;
int **npaths_return*;

XFreeFontPath(*list*)
char ***list*;

ARGUMENTS

directories	Specifies the directory path used to look for a font. Setting the path to the empty list restores the default path defined for the X server.
display	Specifies the connection to the X server.
list	Specifies the array of strings you want to free.
ndirs	Specifies the number of directories in the path.
npaths_return	Returns the number of strings in the font path array.

DESCRIPTION

The **XSetFontPath** function defines the directory search path for font lookup. There is only one search path per X server, not one per client. The interpretation of the strings is operating system dependent, but they are intended to specify directories to be searched in the order listed. Also, the contents of these strings are operating system dependent and are not intended to be used by client applications. Usually, the X server is free to cache font information internally rather than having to read fonts from files. In addition, the X server is guaranteed to flush all cached information about fonts for which there currently are no explicit resource IDs allocated. The meaning of an error from this request is operating system dependent.

XSetFontPath can generate a **BadValue** error.

The **XGetFontPath** function allocates and returns an array of strings containing the search path. When it is no longer needed, the data in the font path should be freed by using **XFreeFontPath**.

The **XFreeFontPath** function frees the data allocated by **XGetFontPath**.

DIAGNOSTICS

BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XListFont(3), XLoadFonts(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetInputFocus, XGetInputFocus – control input focus

SYNTAX

XSetInputFocus(*display, focus, revert_to, time*)
Display **display*;
Window *focus*;
int *revert_to*;
Time *time*;

XGetInputFocus(*display, focus_return, revert_to_return*)
Display **display*;
Window **focus_return*;
int **revert_to_return*;

ARGUMENTS

display Specifies the connection to the X server.

focus Specifies the window, **PointerRoot**, or **None**.

focus_return Returns the focus window, **PointerRoot**, or **None**.

revert_to Specifies where the input focus reverts to if the window becomes not viewable. You can pass **RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**.

revert_to_return
Returns the current focus state (**RevertToParent**, **RevertToPointerRoot**, or **RevertToNone**).

time Specifies the time. You can pass either a timestamp or **CurrentTime**.

DESCRIPTION

The **XSetInputFocus** function changes the input focus and the last-focus-change time. It has no effect if the specified time is earlier than the current last-focus-change time or is later than the current X server time. Otherwise, the last-focus-change time is set to the specified time (**CurrentTime** is replaced by the current X server time). **XSetInputFocus** causes the X server to generate **FocusIn** and **FocusOut** events.

Depending on the focus argument, the following occurs:

- If focus is **None**, all keyboard events are discarded until a new focus window is set, and the revert_to argument is ignored.
- If focus is a window, it becomes the keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported as usual. Otherwise, the event is reported relative to the focus window.
- If focus is **PointerRoot**, the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, the revert_to argument is ignored.

The specified focus window must be viewable at the time **XSetInputFocus** is called, or a **BadMatch** error results. If the focus window later becomes not viewable, the X server evaluates the revert_to argument to determine the new focus window as follows:

- If revert_to is **RevertToParent**, the focus reverts to the parent (or the closest viewable ancestor), and the new revert_to value is taken to be **RevertToNone**.
- If revert_to is **RevertToPointerRoot** or **RevertToNone**, the focus reverts to **PointerRoot** or **None**, respectively. When the focus reverts, the X server generates **FocusIn** and **FocusOut** events, but the last-focus-change time is not affected.

XSetInputFocus can generate **BadMatch**, **BadValue**, and **BadWindow** errors.

The **XGetInputFocus** function returns the focus window and the current focus state.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XWarpPointer(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetLineAttribute, XSetDashes – GC convenience routines

SYNTAX

XSetLineAttributes(*display, gc, line_width, line_style, cap_style, join_style*)
Display **display*;
GC *gc*;
unsigned int *line_width*;
int *line_style*;
int *cap_style*;
int *join_style*;

XSetDashes(*display, gc, dash_offset, dash_list, n*)
Display **display*;
GC *gc*;
int *dash_offset*;
char *dash_list*[];
int *n*;

ARGUMENTS

cap_style	Specifies the line-style and cap-style you want to set for the specified GC. You can pass **CapNotLast**, **CapButt**, **CapRound**, or **CapProjecting**.
dash_list	Specifies the dash-list for the dashed line-style you want to set for the specified GC.
dash_offset	Specifies the phase of the pattern for the dashed line-style you want to set for the specified GC.
display	Specifies the connection to the X server.
gc	Specifies the GC.
join_style	Specifies the line join-style you want to set for the specified GC. You can pass **JoinMiter**, **JoinRound**, or **JoinBevel**.
line_style	Specifies the line-style you want to set for the specified GC. You can pass **LineSolid**, **LineOnOffDash**, or **LineDoubleDash**.
line_width	Specifies the line-width you want to set for the specified GC.
n	Specifies the number of elements in dash_list.

DESCRIPTION

The **XSetLineAttributes** function sets the line drawing components in the specified GC.

XSetLineAttributes can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

The **XSetDashes** function sets the dash-offset and dash-list attributes for dashed line styles in the specified GC. There must be at least one element in the specified dash_list, or a **BadValue** error results. The initial and alternating elements (second, fourth, and so on) of the dash_list are the even dashes, and the others are the odd dashes. Each element specifies a dash length in pixels. All of the elements must be nonzero, or a **BadValue** error results. Specifying an odd-length list is equivalent to specifying the same list concatenated with itself to produce an even-length list.

The dash-offset defines the phase of the pattern, specifying how many pixels into the dash-list the pattern should actually begin in any single graphics request. Dashing is continuous through path elements combined with a join-style but is reset to the dash-offset between each sequence of joined lines.

The unit of measure for dashes is the same for the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but implementations are only required to match this ideal for horizontal and vertical lines. Failing the ideal semantics, it is suggested that the length be measured along the major axis of the line. The major axis is defined as the x axis for lines drawn at an angle of between –45 and +45 degrees or between 315 and 225 degrees from the x axis. For all other lines, the major axis is the y axis.

XSetDashes can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadGC A value for a GContext argument does not name a defined GContext.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCreateGC(3), XQueryBestSize(3), XSetArcMode(3), XSetClipOrigin(3), XSetFillStyle(3), XSetFont(3), XSetState(3), XSetTile(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetPointerMapping, XGetPointerMapping - manipulate pointer settings

SYNTAX

int XSetPointerMapping(*display, map, nmap*)
Display **display*;
unsigned char *map*[];
int *nmap*;

int XGetPointerMapping(*display, map_return, nmap*)
Display **display*;
unsigned char *map_return*[];
int *nmap*;

ARGUMENTS

display	Specifies the connection to the X server.
map	Specifies the mapping list.
map_return	Returns the mapping list.
nmap	Specifies the number of items in the mapping list.

DESCRIPTION

The **XSetPointerMapping** function sets the mapping of the pointer. If it succeeds, the X server generates a **MappingNotify** event, and **XSetPointerMapping** returns **MappingSuccess**. Element map[i] defines the logical button number for the physical button i+1. The length of the list must be the same as **XGetPointerMapping** would return, or a **BadValue** error results. A zero element disables a button, and elements are not restricted in value by the number of physical buttons. However, no two elements can have the same nonzero value, or a **BadValue** error results. If any of the buttons to be altered are logically in the down state, **XSetPointerMapping** returns **MappingBusy**, and the mapping is not changed.

XSetPointerMapping can generate a **BadValue** error.

The **XGetPointerMapping** function returns the current mapping of the pointer. Pointer buttons are numbered starting from one. **XGetPointerMapping** returns the number of physical buttons actually on the pointer. The nominal mapping for a pointer is map[i]=i+1. The nmap argument specifies the length of the array where the pointer mapping is returned, and only the first nmap elements are returned in map_return.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XChangeKeyboardControl(3), XChangeKeyboardMapping(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetScreenSaver, XForceScreenSaver, XActivateScreenSaver, XResetScreenSaver, XGetScreenSaver – manipulate the screen saver

SYNTAX

XSetScreenSaver(*display, timeout, interval, prefer_blanking, allow_exposures*)
Display **display*;
int *timeout, interval*;
int *prefer_blanking*;
int *allow_exposures*;

XForceScreenSaver(*display, mode*)
Display **display*;
int *mode*;

XActivateScreenSaver(*display*)
Display **display*;

XResetScreenSaver(*display*)
Display **display*;

XGetScreenSaver(*display, timeout_return, interval_return, prefer_blanking_return, allow_exposures_return*)
Display **display*;
int **timeout_return*, **interval_return*;
int **prefer_blanking_return*;
int **allow_exposures_return*;

ARGUMENTS

allow_exposures Specifies the screen save control values. You can pass **DontAllowExposures**, **AllowExposures**, or **DefaultExposures**.

allow_exposures_return
Returns the current screen save control value (**DontAllowExposures**, **AllowExposures**, or **DefaultExposures**).

display Specifies the connection to the X server.

interval Specifies the interval, in seconds, between screen saver alterations.

interval_return Returns the interval between screen saver invocations.

mode Specifies the mode that is to be applied. You can pass **ScreenSaverActive** or **ScreenSaverReset**.

prefer_blanking Specifies how to enable screen blanking. You can pass **DontPreferBlanking**, **PreferBlanking**, or **DefaultBlanking**.

prefer_blanking_return
Returns the current screen blanking preference (**DontPreferBlanking**, **PreferBlanking**, or **DefaultBlanking**).

timeout Specifies the timeout, in seconds, until the screen saver turns on.

timeout_return Returns the timeout, in seconds, until the screen saver turns on.

DESCRIPTION

Timeout and interval are specified in seconds. A timeout of 0 disables the screen saver (but an activated screen saver is not deactivated), and a timeout of –1 restores the default. Other negative values generate a **BadValue** error. If the timeout value is nonzero, **XSetScreenSaver** enables the screen saver. An interval of 0 disables the random-pattern motion. If no input from devices (keyboard, mouse, and so on) is generated for the specified number of timeout seconds once the screen saver is enabled, the screen saver is activated.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen simply goes blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending **Expose** events to clients, the screen is tiled with the root window background tile randomly re-origined each interval minutes. Otherwise, the screens' state do not change, and the screen saver is not activated. The screen saver is deactivated, and all screen states are restored at the next keyboard or pointer input or at the next call to **XForceScreenSaver** with mode **ScreenSaverReset**.

If the server-dependent screen saver method supports periodic change, the interval argument serves as a hint about how long the change period should be, and zero hints that no periodic change should be made. Examples of ways to change the screen include scrambling the colormap periodically, moving an icon image around the screen periodically, or tiling the screen with the root window background tile, randomly re-origined periodically.

XSetScreenSaver can generate a **BadValue** error.

If the specified mode is **ScreenSaverActive** and the screen saver currently is deactivated, **XForceScreenSaver** activates the screen saver even if the screen saver had been disabled with a timeout of zero. If the specified mode is **ScreenSaverReset** and the screen saver currently is enabled, **XForceScreenSaver** deactivates the screen saver if it was activated, and the activation timer is reset to its initial state (as if device input had been received).

XForceScreenSaver can generate a **BadValue** error.

The **XActivateScreenSaver** function activates the screen saver.

The **XResetScreenSaver** function resets the screen saver.

The **XGetScreenSaver** function gets the current screen saver values.

DIAGNOSTICS

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetSelectionOwner, XGetSelectionOwner, XConvertSelection – manipulate window selection

SYNTAX

XSetSelectionOwner(*display, selection, owner, time*)
Display **display*;
Atom *selection*;
Window *owner*;
Time *time*;

Window XGetSelectionOwner(*display, selection*)
Display **display*;
Atom *selection*;

XConvertSelection(*display, selection, target, property, requestor, time*)
Display **display*;
Atom *selection, target*;
Atom *property*;
Window *requestor*;
Time *time*;

ARGUMENTS

display	Specifies the connection to the X server.
owner	Specifies the owner of the specified selection atom. You can pass a window or **None**.
property	Specifies the property name. You also can pass **None**.
requestor	Specifies the requestor.
selection	Specifies the selection atom.
target	Specifies the target atom.
time	Specifies the time. You can pass either a timestamp or **CurrentTime**.

DESCRIPTION

The **XSetSelectionOwner** function changes the owner and last-change time for the specified selection and has no effect if the specified time is earlier than the current last-change time of the specified selection or is later than the current X server time. Otherwise, the last-change time is set to the specified time, with **CurrentTime** replaced by the current server time. If the owner window is specified as **None**, then the owner of the selection becomes **None** (that is, no owner). Otherwise, the owner of the selection becomes the client executing the request.

If the new owner (whether a client or **None**) is not the same as the current owner of the selection and the current owner is not **None**, the current owner is sent a **SelectionClear** event. If the client that is the owner of a selection is later terminated (that is, its connection is closed) or if the owner window it has specified in the request is later destroyed, the owner of the selection automatically reverts to **None**, but the last-change time is not affected. The selection atom is uninterpreted by the X server. **XGetSelectionOwner** returns the owner window, which is reported in **SelectionRequest** and **SelectionClear** events. Selections are global to the X server.

XSetSelectionOwner can generate **BadAtom** and **BadWindow** errors.

The **XGetSelectionOwner** function returns the window ID associated with the window that currently owns the specified selection. If no selection was specified, the function returns the constant **None**. If **None** is returned, there is no owner for the selection.

XGetSelectionOwner can generate a **BadAtom** error.

XConvertSelection requests that the specified selection be converted to the specified target type:

- If the specified selection has an owner, the X server sends a **SelectionRequest** event to that owner.
- If no owner for the specified selection exists, the X server generates a **SelectionNotify** event to the requestor with property **None**.

The arguments are passed on unchanged in either of the events. There are two predefined selection atoms: PRIMARY and SECONDARY.

XConvertSelection can generate **BadAtom** and **BadWindow** errors.

DIAGNOSTICS

BadAtom A value for an Atom argument does not name a defined Atom.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetState, XSetFunction, XSetPlanemask, XSetForeground, XSetBackground – GC convenience routines

SYNTAX

XSetState(*display, gc, foreground, background, function, plane_mask*)
Display **display*;
GC *gc*;
unsigned long *foreground, background*;
int *function*;
unsigned long *plane_mask*;

XSetFunction(*display, gc, function*)
Display **display*;
GC *gc*;
int *function*;

XSetPlaneMask(*display, gc, plane_mask*)
Display **display*;
GC *gc*;
unsigned long *plane_mask*;

XSetForeground(*display, gc, foreground*)
Display **display*;
GC *gc*;
unsigned long *foreground*;

XSetBackground(*display, gc, background*)
Display **display*;
GC *gc*;
unsigned long *background*;

ARGUMENTS

background	Specifies the background you want to set for the specified GC.
display	Specifies the connection to the X server.
foreground	Specifies the foreground you want to set for the specified GC.
function	Specifies the function you want to set for the specified GC.
gc	Specifies the GC.
plane_mask	Specifies the plane mask.

DESCRIPTION

The **XSetState** function sets the foreground, background, plane mask, and function components for the specified GC.

XSetState can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

XSetFunction sets a specified value in the specified GC.

XSetFunction can generate **BadAlloc**, **BadGC**, and **BadValue** errors.

The **XSetPlaneMask** function sets the plane mask in the specified GC.

XSetPlaneMask can generate **BadAlloc** and **BadGC** errors.

The **XSetForeground** function sets the foreground in the specified GC.

XSetForeground can generate **BadAlloc** and **BadGC** errors.

The **XSetBackground** function sets the background in the specified GC.

XSetBackground can generate **BadAlloc** and **BadGC** errors.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadGC A value for a GContext argument does not name a defined GContext.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XCreateGC(3), XQueryBestSize(3), XSetArcMode(3), XSetClipOrigin(3), XSetFillStyle(3), XSetFont(3), XSetLineAttributes(3), XSetTile(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetTextProperty, XGetTextProperty – set and read text properties

SYNTAX

void XSetTextProperty(*display, w, text_prop, property*)
Display **display*;
Window *w*;
XTextProperty **text_prop*;
Atom *property*;

Status XGetTextProperty(*display, w, text_prop_return, property*)
Display **display*;
Window *w*;
XTextProperty **text_prop_return*;
Atom *property*;

ARGUMENTS

display Specifies the connection to the X server.

property Specifies the property name.

text_prop Specifies the **XTextProperty** structure to be used.

text_prop_return
Returns the **XTextProperty** structure.

DESCRIPTION

The **XSetTextProperty** function replaces the existing, specified property for the named window with the data, type, format, and number of items determined by the value field, the encoding field, the format field, and the nitems field, respectively, of the specified **XTextProperty** structure. If the property does not already exist, **XSetTextProperty** sets it for the specified window.

XSetTextProperty can generate **BadAlloc**, **BadAtom**, **BadValue**, and **BadWindow** errors.

The **XGetTextProperty** function reads the specified property from the window and stores the data in the returned **XTextProperty** structure. It stores the data in the value field, the type of the data in the encoding field. the format of the data in the format field, and the number of items of data in the nitems field. The particular interpretation of the property's encoding and data as "text" is left to the calling application. If the specified property does not exist on the window, **XGetTextProperty** sets the value field to NULL, the encoding field to None, the format field to 0, and the nitems field to zero.

If it was able to set these files in the **XTextProperty** structure, **XGetTextProperty** returns a non-zero status; otherwise, it returns a zero status.

XGetTextProperty can generate **BadAtom** and **BadWindow** errors.

PROPERTIES

WM_CLIENT_MACHINE
The string name of the machine on which the client application is running.

WM_COMMAND
The command and arguments, separated by ASCII nulls, used to invoke the application.

WM_ICON_NAME
Name to be used in icon.

WM_NAME Name of the application.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadAtom A value for an Atom argument does not name a defined Atom.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XSetCommand(3), XSetTransientForHint(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetTile, XSetStipple, XSetTSOrigin - GC convenience routines

SYNTAX

XSetTile(*display, gc, tile*)
Display **display*;
GC *gc*;
Pixmap *tile*;

XSetStipple(*display, gc, stipple*)
Display **display*;
GC *gc*;
Pixmap *stipple*;

XSetTSOrigin(*display, gc, ts_x_origin, ts_y_origin*)
Display **display*;
GC *gc*;
int *ts_x_origin, ts_y_origin*;

ARGUMENTS

display	Specifies the connection to the X server.
gc	Specifies the GC.
stipple	Specifies the stipple you want to set for the specified GC.
tile	Specifies the fill tile you want to set for the specified GC.
ts_x_origin *ts_y_origin*	Specify the x and y coordinates of the tile and stipple origin.

DESCRIPTION

The **XSetTile** function sets the fill tile in the specified GC. The tile and GC must have the same depth, or a **BadMatch** error results.

XSetTile can generate **BadAlloc**, **BadGC**, **BadMatch**, and **BadPixmap** errors.

The **XSetStipple** function sets the stipple in the specified GC. The stipple and GC must have the same depth, or a **BadMatch** error results.

XSetStipple can generate **BadAlloc**, **BadGC**, **BadMatch**, and **BadPixmap** errors.

The **XSetTSOrigin** function sets the tile/stipple origin in the specified GC. When graphics requests call for tiling or stippling, the parent's origin will be interpreted relative to whatever destination drawable is specified in the graphics request.

XSetTSOrigin can generate **BadAlloc** and **BadGC** errors.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadGC	A value for a GContext argument does not name a defined GContext.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadPixmap	A value for a Pixmap argument does not name a defined Pixmap.

SEE ALSO

XCreateGC(3), XQueryBestSize(3), XSetArcMode(3), XSetClipOrigin(3), XSetFillStyle(3), XSetFont(3), XSetLineAttributes(3), XSetState(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales

representative.

NAME

XSetTransientForHint, XGetTransientForHint - set or read a window's WM_TRANSIENT_FOR property

SYNTAX

XSetTransientForHint(*display, w, prop_window*)
Display **display*;
Window *w*;
Window *prop_window*;

Status XGetTransientForHint(*display, w, prop_window_return*)
Display **display*;
Window *w*;
Window **prop_window_return*;

ARGUMENTS

display Specifies the connection to the X server.

w Specifies the window.

prop_window Specifies the window that the WM_TRANSIENT_FOR property is to be set to.

prop_window_return
Returns the WM_TRANSIENT_FOR property of the specified window.

DESCRIPTION

The **XSetTransientForHint** function sets the WM_TRANSIENT_FOR property of the specified window to the specified prop_window.

XSetTransientForHint can generate **BadAlloc** and **BadWindow** errors.

The **XGetTransientForHint** function returns the WM_TRANSIENT_FOR property for the specified window.

XGetTransientForHint can generate a **BadWindow** error.

PROPERTIES

WM_TRANSIENT_FOR
Set by application programs to indicate to the window manager that a transient top-level window, such as a dialog box.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XSetCommand(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetWMClientMachine, XGetWMClientMachine – set or read a window's WM_CLIENT_MACHINE property

SYNTAX

void XSetWMClientMachine(*display, w, text_prop*)
Display **display*;
Window *w*;
XTextProperty **text_prop*;

Status XGetWMClientMachine(*display, w, text_prop_return*)
Display **display*;
Window *w*;
XTextProperty **text_prop_return*;

ARGUMENTS

display Specifies the connection to the X server.

text_prop Specifies the **XTextProperty** structure to be used.

text_prop_return
Returns the **XTextProperty** structure.

w Specifies the window.

DESCRIPTION

The **XSetWMClientMachine** convenience function performs a **XSetTextProperty** on the WM_CLIENT_MACHINE property. Note that you also can set the client machine property by using **XSetTextProperty** (see section 9.1.3).

The **XGetWMClientMachine** convenience function performs an **XGetTextProperty** on the WM_CLIENT_MACHINE property. Note that you also can read the client machine property by using **XGetTextProperty** (see section 9.1.3).

PROPERTIES

WM_CLIENT_MACHINE
The string name of the machine on which the client application is running.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetWMColormapWindows, XGetWMColormapWindows – set or read a window's WM_COLORMAP_WINDOWS property

SYNTAX

Status XSetWMColormapWindows(*display, w, colormap_windows, count*)
Display **display*;
Window *w*;
Window **colormap_windows*;
int *count*;

Status XGetWMColormapWindows(*display, w, colormap_windows_return, count_return*)
Display **display*;
Window *w*;
Window ***colormap_windows_return*;
int **count_return*;

ARGUMENTS

display Specifies the connection to the X server.

colormap_windows
Specifies the list of windows.

colormap_windows_return
Returns the list of windows.

count Specifies the number of windows in the list.

count_return Returns the number of windows in the list.

w Specifies the window.

DESCRIPTION

The **XSetWMColormapWindows** function replaces the WM_COLORMAP_WINDOWS property on the specified window with the list of windows specified by the colormap_windows argument. It the property does not already exist, **XSetWMColormapWindows** sets the WM_COLORMAP_WINDOWS property on the specified window to the list of windows specified by the colormap_windows argument. The property is stored with a type of WINDOW and a format of 32. If it cannot intern the WM_COLORMAP_WINDOWS atom, **XSetWMColormapWindows** returns a zero status. Otherwise, it returns a non-zero status.

XSetWMColormapWindows can generate **BadAlloc** and **BadWindow** errors.

The **XGetWMColormapWindows** function returns the list of window identifiers stored in the WM_COLORMAP_WINDOWS property on the specified window. These identifiers indicate the colormaps that the window manager may need to install for this window. If the property exists, is of type WINDOW, is of format 32, and the atom WM_COLORMAP_WINDOWS can be interned, **XGetWMColormapWindows** sets the windows_return argument to a list of window identifiers, sets the count_return argument to the number of elements in list, and returns a nonzero status. Otherwise, it sets neither of the return arguments and returns a zero status. To release the list of window identifiers, use **XFree**.

XGetWMColormapWindows can generate a **BadWindow** error.

PROPERTIES

WM_COLORMAP_WINDOWS
List of window IDs that may need a different colormap than that of their top-level window.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetWMIconName, XGetWMIconName, XSetIconName, XGetIconName – set or read a window's WM_ICON_NAME property

SYNTAX

void XSetWMIconName(*display, w, text_prop*)
Display **display*;
Window *w*;
XTextProperty **text_prop*;

Status XGetWMIconName(*display, w, text_prop_return*)
Display **display*;
Window *w*;
XTextProperty **text_prop_return*;

XSetIconName(*display, w, icon_name*)
Display **display*;
Window *w*;
char **icon_name*;

Status XGetIconName(*display, w, icon_name_return*)
Display **display*;
Window *w*;
char ***icon_name_return*;

ARGUMENTS

display Specifies the connection to the X server.

icon_name Specifies the icon name, which should be a null-terminated string.

icon_name_return
Returns a pointer to the window's icon name, which is a null-terminated string.

text_prop Specifies the **XTextProperty** structure to be used.

text_prop_return
Returns the **XTextProperty** structure.

w Specifies the window.

DESCRIPTION

The **XSetWMIconName** convenience function performs a **XSetTextProperty** on the WM_ICON_NAME property (see section 9.1.3).

The **XGetWMIconName** convenience function performs an **XGetTextProperty** on the WM_ICON_NAME property (see section 9.1.3).

The **XSetIconName** function sets the name to be displayed in a window's icon.

XSetIconName can generate **BadAlloc** and **BadWindow** errors.

The **XGetIconName** function returns the name to be displayed in the specified window's icon. If it succeeds, it returns nonzero; otherwise, if no icon name has been set for the window, it returns zero. If you never assigned a name to the window, **XGetIconName** sets icon_name_return to NULL. When finished with it, a client must free the icon name string using **XFree**.

XGetIconName can generate a **BadWindow** error.

PROPERTIES

WM_ICON_NAME
Name to be used in icon.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetWMName, XGetWMName, XStoreName, XFetchName – set or read a window's WM_NAME property

SYNTAX

void XSetWMName(*display, w, text_prop*)
Display **display*;
Window *w*;
XTextProperty **text_prop*;

Status XGetWMName(*display, w, text_prop_return*)
Display **display*;
Window *w*;
XTextProperty **text_prop_return*;

XStoreName(*display, w, window_name*)
Display **display*;
Window *w*;
char **window_name*;

Status XFetchName(*display, w, window_name_return*)
Display **display*;
Window *w*;
char ***window_name_return*;

ARGUMENTS

display Specifies the connection to the X server.

text_prop Specifies the **XTextProperty** structure to be used.

text_prop_return
Returns the **XTextProperty** structure.

w Specifies the window.

window_name Specifies the window name, which should be a null-terminated string.

window_name_return
Returns a pointer to the window name, which is a null-terminated string.

DESCRIPTION

The **XSetWMName** convenience function performs a **XSetTextProperty** on the WM_NAME property (see section 9.1.3).

The **XGetWMName** convenience function performs an **XGetTextProperty** on the WM_NAME property (see section 9.1.3).

The **XStoreName** function assigns the name passed to window_name to the specified window. A window manager can display the window name in some prominent place, such as the title bar, to allow users to identify windows easily. Some window managers may display a window's name in the window's icon, although they are encouraged to use the window's icon name if one is provided by the application.

XStoreName can generate **BadAlloc** and **BadWindow** errors.

The **XFetchName** function returns the name of the specified window. If it succeeds, it returns nonzero; otherwise, no name has been set for the window, and it returns zero. If the WM_NAME property has not been set for this window, **XFetchName** sets window_name_return to NULL. When finished with it, a client must free the window name string using **XFree**.

XFetchName can generate a **BadWindow** error.

PROPERTIES

WM_NAME Name of the application.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMProperties(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetWMProperties, XWMGeometry - window manager convenience functions

SYNTAX

void XSetWMProperties(*display, w, window_name, icon_name, argv, argc, normal_hints, wm_hints, class_hints*)
Display **display*;
Window *w*;
XTextProperty **window_name*;
XTextProperty **icon_name*;
char ***argv*;
int *argc*;
XSize_hints **normal_hints*;
XWMHints **wm_hints*;
XClassHint **class_hints*;

int XWMGeometry(*display, screen, user_geom, def_geom, bwidth, hints, x_return, y_return, width_return, height_return, gravity_return*)
Display **display*;
int *screen*;
char **user_geom*;
char **def_geom*;
unsigned int *bwidth*;
XSizeHints **hints*;
int **x_return*, **y_return*;
int **width_return*;
int **height_return*;
int **gravity_return*;

ARGUMENTS

argc	Specifies the number of arguments.
argv	Specifies the application's argument list.
bwidth	Specifies the border width.
class_hints	Specifies the **XClassHint** structure to be used.
def_geom	Specifies the application's default geometry or NULL.
display	Specifies the connection to the X server.
gravity_return	Returns the window gravity.
hints	Specifies the size hints for the window in its normal state.
icon_name	Specifies the icon name, which should be a null-terminated string.
normal_hints	Specifies the size hints for the window in its normal state.
screen	Specifies the screen.
user_geom	Specifies the user-specified geometry or NULL.
x_return *y_return*	Return the x and y offsets.
width_return *height_return*	Return the width and height determined.
w	Specifies the window.
window_name	Specifies the window name, which should be a null-terminated string.
wm_hints	Specifies the **XWMHints** structure to be used.

DESCRIPTION

The **XSetWMProperties** convenience function provides a single programming interface for setting those essential window properties that are used for communicating with other clients (particularly window and session managers).

If the window_name argument is non-null, **XSetWMProperties** calls **XSetWMName**, which, in turn, sets the WM_NAME property (see section 9.1.4). If the icon_name argument is non-null, **XSetWMProperties** calls **XSetWMIconName**, which sets the WM_ICON_NAME property (see section 9.1.5). If the argv argument is non-null, **XSetWMProperties** calls **XSetCommand**, which sets the WM_COMMAND property (see section 9.2.1). Note that an argc of 0 is allowed to indicate a zero-length command. Note also that the hostname of this machine is stored using **XSetWMClientMachine** (see section 9.2.2).

If the normal_hints argument is non-null, **XSetWMProperties** calls **XSetWMNormalHints**, which sets the WM_NORMAL_HINTS property (see section 9.1.7). If the wm_hints argument is non-null, **XSetWMProperties** calls **XSetWMHints**, which sets the WM_HINTS property (see section 9.1.6).

If the class_hints argument is non-null, **XSetWMProperties** calls **XSetClassHint**, which sets the WM_CLASS property (see section 9.1.8). If the res_name member in the **XClassHint** structure is set to the null pointer and the RESOURCE_NAME environment variable is set, then value of the environment variable is substituted for res_name. If the res_name member is null, the environment variable is not set, and argv and argv[0] are set, then the value of argv[0], stripped of any directory prefixes, is substituted for res_name.

XSetWMProperties can generate **BadAlloc** and **BadWindow** errors.

The **XWMGeometry** function combines any geometry information (given in the format used by **XParseGeometry**) specified by the user and by the calling program with size hints (usually the ones to be stored in WM_NORMAL_HINTS) and returns the position, size, and gravity (**NorthWestGravity**, **NorthEastGravity**, **SouthEastGravity** or **SouthWestGravity**) that describe the window. If the base size is not set in the **XSizeHints** structure, the minimum size is used if set. Otherwise, a base size of 0 is assumed. If no minimum size is set in the hints structure, the base size is used. A mask (in the form returned by **XParseGeometry**) that describes which values came from the user specification and whether or not the position coordinates are relative to the right and bottom edges is returned (which will have already been accounted for in the x_return and y_return values).

Note that invalid geometry specifications can cause a width or height of 0 to be returned. The caller may pass the address of the hints win_gravity field as gravity_return to update the hints directly.

PROPERTIES

WM_CLASS Set by application programs to allow window and session managers to obtain the application's resources from the resource database.

WM_CLIENT_MACHINE
The string name of the machine on which the client application is running.

WM_COMMAND
The command and arguments, separated by ASCII nulls, used to invoke the application.

WM_HINTS Additional hints set by client for use by the window manager. The C type of this property is **XWMHints**.

WM_ICON_NAME
Name to be used in icon.

WM_NAME Name of the application.

WM_NORMAL_HINTS
Size hints for a window in its normal state. The C type of this property is **XSizeHints**.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XParseGeometry(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProtocols(3), XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSetWMProtocols, XGetWMProtocols – set or read a window's WM_PROTOCOLS property

SYNTAX

Status XSetWMProtocols(*display, w, protocols, count*)
Display **display*;
Window *w*;
Atom **protocols*;
int *count*;

Status XGetWMProtocols(*display, w, protocols_return, count_return*)
Display **display*;
Window *w*;
Atom ***protocols_return*;
int **count_return*;

ARGUMENTS

display Specifies the connection to the X server.

count Specifies the number of protocols in the list.

count_return Returns the number of protocols in the list.

protocols Specifies the list of protocols.

protocols_return
Returns the list of protocols.

DESCRIPTION

The **XSetWMProtocols** function replaces the WM_PROTOCOLS property on the specified window with the list of atoms specified by the protocols argument. If the property does not already exist, **XSetWMProtocols** sets the WM_PROTOCOLS property on the specified window to the list of atoms specified by the protocols argument. The property is stored with a type of ATOM and a format of 32. If it cannot intern the WM_PROTOCOLS atom, **XSetWMProtocols** returns a zero status. Otherwise, it returns a non-zero status.

XSetWMProtocols can generate **BadAlloc** and **BadWindow** errors.

The **XGetWMProtocols** function returns the list of atoms stored in the WM_PROTOCOLS property on the specified window. These atoms describe window manager protocols in which the owner of this window is willing to participate. If the property exists, is of type ATOM, is of format 32, and the atom WM_PROTOCOLS can be interned, **XGetWMProtocols** sets the protocols_return argument to a list of atoms, sets the count_return argument to the number of elements in list, and returns a a non-zero status. Otherwise, it sets neither of the return arguments and returns a zero status. To release the list of atom, use **XFree**.

XGetWMProtocols can generate a **BadWindow** error.

PROPERTIES

WM_PROTOCOLS
List of atoms that identify the communications protocols between the client and window manager in which the client is willing to participate.

DIAGNOSTICS

BadAlloc The server failed to allocate the requested resource or server memory.

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3),

XStringListToTextProperty(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XStoreBytes, XStoreBuffer, XFetchBytes, XFetchBuffer, XRotateBuffers – manipulate cut and paste buffers

SYNTAX

XStoreBytes(*display, bytes, nbytes*)
Display **display*;
char **bytes*;
int *nbytes*;

XStoreBuffer(*display, bytes, nbytes, buffer*)
Display **display*;
char **bytes*;
int *nbytes*;
int *buffer*;

char *XFetchBytes(*display, nbytes_return*)
Display **display*;
int **nbytes_return*;

char *XFetchBuffer(*display, nbytes_return, buffer*)
Display **display*;
int **nbytes_return*;
int *buffer*;

XRotateBuffers(*display, rotate*)
Display **display*;
int *rotate*;

ARGUMENTS

buffer	Specifies the buffer in which you want to store the bytes or from which you want the stored data returned.
bytes	Specifies the bytes, which are not necessarily ASCII or null-terminated.
display	Specifies the connection to the X server.
nbytes	Specifies the number of bytes to be stored.
nbytes_return	Returns the number of bytes in the buffer.
rotate	Specifies how much to rotate the cut buffers.

DESCRIPTION

Note that the cut buffer's contents need not be text, so zero bytes are not special. The cut buffer's contents can be retrieved later by any client calling **XFetchBytes**.

XStoreBytes can generate a **BadAlloc** error.

If the property for the buffer has never been created, a **BadAtom** error results.

XStoreBuffer can generate **BadAlloc** and **BadAtom** errors.

The **XFetchBytes** function returns the number of bytes in the nbytes_return argument, if the buffer contains data. Otherwise, the function returns NULL and sets nbytes to 0. The appropriate amount of storage is allocated and the pointer returned. The client must free this storage when finished with it by calling **XFree**. Note that the cut buffer does not necessarily contain text, so it may contain embedded zero bytes and may not terminate with a null byte.

The **XFetchBuffer** function returns zero to the nbytes_return argument if there is no data in the buffer.

XFetchBuffer can generate a **BadValue** error.

The **XRotateBuffers** function rotates the cut buffers, such that buffer 0 becomes buffer n, buffer 1 becomes n + 1 mod 8, and so on. This cut buffer numbering is global to the display. Note that **XRotateBuffers** generates **BadMatch** errors if any of the eight buffers have not been created.

XRotateBuffers can generate a **BadMatch** error.

DIAGNOSTICS

BadAlloc	The server failed to allocate the requested resource or server memory.
BadAtom	A value for an Atom argument does not name a defined Atom.
BadMatch	Some argument or pair of arguments has the correct type and range but fails to match in some other way required by the request.
BadValue	Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XFree(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XStoreColors, XStoreColor, XStoreNamedColor – set colors

SYNTAX

XStoreColors(*display, colormap, color, ncolors*)
Display **display*;
Colormap *colormap*;
XColor *color*[];
int *ncolors*;

XStoreColor(*display, colormap, color*)
Display **display*;
Colormap *colormap*;
XColor **color*;

XStoreNamedColor(*display, colormap, color, pixel, flags*)
Display **display*;
Colormap *colormap*;
char **color*;
unsigned long *pixel*;
int *flags*;

ARGUMENTS

color	Specifies the pixel and RGB values or the color name string (for example, red).
color	Specifies an array of color definition structures to be stored.
colormap	Specifies the colormap.
display	Specifies the connection to the X server.
flags	Specifies which red, green, and blue components are set.
ncolors	Specifies the number of **XColor** structures in the color definition array.
pixel	Specifies the entry in the colormap.

DESCRIPTION

The **XStoreColors** function changes the colormap entries of the pixel values specified in the pixel members of the **XColor** structures. You specify which color components are to be changed by setting **DoRed**, **DoGreen**, and/or **DoBlue** in the flags member of the **XColor** structures. If the colormap is an installed map for its screen, the changes are visible immediately. **XStoreColors** changes the specified pixels if they are allocated writable in the colormap by any client, even if one or more pixels generates an error. If a specified pixel is not a valid index into the colormap, a **BadValue** error results. If a specified pixel either is unallocated or is allocated read-only, a **BadAccess** error results. If more than one pixel is in error, the one that gets reported is arbitrary.

XStoreColors can generate **BadAccess**, **BadColor**, and **BadValue** errors.

The **XStoreColor** function changes the colormap entry of the pixel value specified in the pixel member of the **XColor** structure. You specified this value in the pixel member of the **XColor** structure. This pixel value must be a read/write cell and a valid index into the colormap. If a specified pixel is not a valid index into the colormap, a **BadValue** error results. **XStoreColor** also changes the red, green, and/or blue color components. You specify which color components are to be changed by setting **DoRed**, **DoGreen**, and/or **DoBlue** in the flags member of the **XColor** structure. If the colormap is an installed map for its screen, the changes are visible immediately.

XStoreColor can generate **BadAccess**, **BadColor**, and **BadValue** errors.

The **XStoreNamedColor** function looks up the named color with respect to the screen associated with the colormap and stores the result in the specified colormap. The pixel argument determines the entry in the colormap. The flags argument determines which of the red, green, and blue components are set. You can set this member to the bitwise inclusive OR of the bits **DoRed**, **DoGreen**, and **DoBlue**. If the specified pixel is not a valid index into the colormap, a **BadValue** error results. If the specified pixel either is unallocated or is allocated read-only, a **BadAccess** error results. You should use the ISO Latin-1 encoding; uppercase and lowercase do not matter.

XStoreNamedColor can generate **BadAccess**, **BadColor**, **BadName**, and **BadValue** errors.

DIAGNOSTICS

BadAccess A client attempted to free a color map entry that it did not already allocate.

BadAccess A client attempted to store into a read-only color map entry.

BadColor A value for a Colormap argument does not name a defined Colormap.

BadName A font or color of the specified name does not exist.

BadValue Some numeric value falls outside the range of values accepted by the request. Unless a specific range is specified for an argument, the full range defined by the argument's type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XAllocColor(3), XCreateColormap(3), XQueryColor(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XStringListToTextProperty, XTextPropertyToStringList, XFreeStringList, XTextProperty - convert string lists and text property structure

SYNTAX

Status XStringListToTextProperty(*list, count, text_prop_return*)
char ***list*;
int *count*;
XTextProperty **text_prop_return*;

Status XTextPropertyToStringList(*text_prop, list_return, count_return*)
XTextProperty **text_prop*;
char ****list_return*;
int **count_return*;

void XFreeStringList(*list*)
char ***list*;

ARGUMENTS

count	Specifies the number of strings.
count_return	Returns the number of strings.
list	Specifies the list of strings to be freed.
list	Specifies a list of null-terminated character strings.
list_return	Returns a list of null-terminated character strings.
text_prop	Specifies the **XTextProperty** structure to be used.
text_prop_return	Returns the **XTextProperty** structure.

DESCRIPTION

The **XStringListToTextProperty** function sets the specified **XTextProperty** to be of type STRING (format 8) with a value representing the concatenation of the specified list of null-separated character strings. An extra byte containing NULL (which is not included in the nitems member) is stored at the end of the value field of text_prop_return. If insufficient memory is available for the new value string, **XStringListToTextProperty** does not set any fields in the **XTextProperty** structure and returns a zero status. Otherwise, it returns a non-zero status. To free the storage for the value field, use **XFree**.

The **XTextPropertyToStringList** function returns a list of strings representing the null-separated elements of the specified **XTextProperty** structure. The data in text_prop must be of type STRING and format 8. Multiple elements of the property (for example, the strings in a disjoint text selection) are separated by a NULL (encoding 0). The contents of the property are not null-terminated. If insufficient memory is available for the list and its elements, **XTextPropertyToStringList** sets no return values and returns a zero status. Otherwise, it returns a non-zero status. To free the storage for the list and its contents, use **XFreeStringList**.

The **XFreeStringList** function releases memory allocated by **XTextPropertyToStringList**.

STRUCTURES

The **XTextProperty** structure contains:

```
typedef struct                  {
        unsigned char *value;   /* property data */
        Atom encoding;          /* type of property */
        int format;             /* 8, 16, or 32 */
        unsigned long nitems;   /* number of items in value */
} XTextProperty;
```

SEE ALSO

XAllocClassHint(3), XAllocIconSize(3), XAllocSizeHints(3), XAllocWMHints(3), XFree(3), XSetCommand(3), XSetTransientForHint(3), XSetTextProperty(3), XSetWMClientMachine(3), XSetWMColormapWindows(3), XSetWMIconName(3), XSetWMName(3), XSetWMProperties(3), XSetWMProtocols(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XStringToKeysym, XKeysymToString, XKeycodeToKeysym, XKeysymToKeycode – convert keysyms

SYNTAX

KeySym XStringToKeysym(*string*)
char **string*;

char *XKeysymToString(*keysym*)
KeySym *keysym*;

KeySym XKeycodeToKeysym(*display, keycode, index*)
Display **display*;
KeyCode *keycode*;
int *index*;

KeyCode XKeysymToKeycode(*display, keysym*)
Display **display*;
KeySym *keysym*;

ARGUMENTS

display	Specifies the connection to the X server.
index	Specifies the element of KeyCode vector.
keycode	Specifies the KeyCode.
keysym	Specifies the KeySym that is to be searched for or converted.
string	Specifies the name of the KeySym that is to be converted.

DESCRIPTION

Valid KeySym names are listed in <**X11/keysymdef.h**> by removing the XK_ prefix from each name. If the specified string does not match a valid KeySym, **XStringToKeysym** returns **NoSymbol**.

The returned string is in a static area and must not be modified. If the specified KeySym is not defined, **XKeysymToString** returns a NULL.

The **XKeycodeToKeysym** function uses internal Xlib tables and returns the KeySym defined for the specified KeyCode and the element of the KeyCode vector. If no symbol is defined, **XKeycodeToKeysym** returns **NoSymbol**.

If the specified KeySym is not defined for any KeyCode, **XKeysymToKeycode** returns zero.

SEE ALSO

XLookupKeysym(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XSynchronize, XSetAfterFunction – enable or disable synchronization

SYNTAX

int (*XSynchronize(*display, onoff*))()
Display **display*;
Bool *onoff*;

int (*XSetAfterFunction(*display, procedure*))()
Display **display*;
int (**procedure*)();

ARGUMENTS

display	Specifies the connection to the X server.
procedure	Specifies the function to be called after an Xlib function that generates a protocol request completes its work.
onoff	Specifies a Boolean value that indicates whether to enable or disable synchronization.

DESCRIPTION

The **XSynchronize** function returns the previous after function. If onoff is **True**, **XSynchronize** turns on synchronous behavior. If onoff is **False**, **XSynchronize** turns off synchronous behavior.

The specified procedure is called with only a display pointer. **XSetAfterFunction** returns the previous after function.

SEE ALSO

XSetErrorHandler(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XTextExtents, XTextExtents16, XQueryTextExtents, XQueryTextExtents16 – compute or query text extents

SYNTAX

XTextExtents(*font_struct, string, nchars, direction_return, font_ascent_return, font_descent_return, overall_return*)

```
XFontStruct *font_struct;
char *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;
```

XTextExtents16(*font_struct, string, nchars, direction_return, font_ascent_return, font_descent_return, overall_return*)

```
XFontStruct *font_struct;
XChar2b *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;
```

XQueryTextExtents(*display, font_ID, string, nchars, direction_return, font_ascent_return, font_descent_return, overall_return*)

```
Display *display;
XID font_ID;
char *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;
```

XQueryTextExtents16(*display, font_ID, string, nchars, direction_return, font_ascent_return, font_descent_return, overall_return*)

```
Display *display;
XID font_ID;
XChar2b *string;
int nchars;
int *direction_return;
int *font_ascent_return, *font_descent_return;
XCharStruct *overall_return;
```

ARGUMENTS

direction_return
Returns the value of the direction hint (**FontLeftToRight** or **FontRightToLeft**).

display Specifies the connection to the X server.

font_ID Specifies either the font ID or the **GContext** ID that contains the font.

font_ascent_return
Returns the font ascent.

font_descent_return
Returns the font descent.

font_struct Specifies a pointer to the **XFontStruct** structure.

nchars Specifies the number of characters in the character string.

string Specifies the character string.

overall_return Returns the overall size in the specified **XCharStruct** structure.

DESCRIPTION

The **XTextExtents** and **XTextExtents16** functions perform the size computation locally and, thereby, avoid the round-trip overhead of **XQueryTextExtents** and **XQueryTextExtents16**. Both functions return an **XCharStruct** structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbearing member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each **XChar2b** structure is interpreted as a 16-bit number with byte1 as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

The **XQueryTextExtents** and **XQueryTextExtents16** functions return the bounding box of the specified 8-bit and 16-bit character string in the specified font or the font contained in the specified GC. These functions query the X server and, therefore, suffer the round-trip overhead that is avoided by **XTextExtents** and **XTextExtents16**. Both functions return a **XCharStruct** structure, whose members are set to the values as follows.

The ascent member is set to the maximum of the ascent metrics of all characters in the string. The descent member is set to the maximum of the descent metrics. The width member is set to the sum of the character-width metrics of all characters in the string. For each character in the string, let W be the sum of the character-width metrics of all characters preceding it in the string. Let L be the left-side-bearing metric of the character plus W. Let R be the right-side-bearing metric of the character plus W. The lbearing member is set to the minimum L of all characters in the string. The rbearing member is set to the maximum R.

For fonts defined with linear indexing rather than 2-byte matrix indexing, each **XChar2b** structure is interpreted as a 16-bit number with byte1 as the most-significant byte. If the font has no defined default character, undefined characters in the string are taken to have all zero metrics.

Characters with all zero metrics are ignored. If the font has no defined default_char, the undefined characters in the string are also ignored.

XQueryTextExtents and **XQueryTextExtents16** can generate **BadFont** and **BadGC** errors.

DIAGNOSTICS

BadFont A value for a Font or GContext argument does not name a defined Font.

BadGC A value for a GContext argument does not name a defined GContext.

SEE ALSO

XLoadFont(3), XTextWidth(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XTextWidth, XTextWidth16 – compute text width

SYNTAX

int XTextWidth(*font_struct, string, count*)
XFontStruct **font_struct*;
char **string*;
int *count*;

int XTextWidth16(*font_struct, string, count*)
XFontStruct **font_struct*;
XChar2b **string*;
int *count*;

ARGUMENTS

count	Specifies the character count in the specified string.
font_struct	Specifies the font used for the width computation.
string	Specifies the character string.

DESCRIPTION

The **XTextWidth** and **XTextWidth16** functions return the width of the specified 8-bit or 2-byte character strings.

SEE ALSO

XLoadFont(3), XTextExtents(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XTranslateCoordinates – translate window coordinates

SYNTAX

Bool XTranslateCoordinates(*display, src_w, dest_w, src_x, src_y, dest_x_return, dest_y_return, child_return*)

Display * *display*;
Window *src_w, dest_w*;
int *src_x, src_y*;
int * *dest_x_return*, * *dest_y_return*;
Window * *child_return*;

ARGUMENTS

child_return	Returns the child if the coordinates are contained in a mapped child of the destination window.
dest_w	Specifies the destination window.
dest_x_return *dest_y_return*	Return the x and y coordinates within the destination window.
display	Specifies the connection to the X server.
src_w	Specifies the source window.
src_x *src_y*	Specify the x and y coordinates within the source window.

DESCRIPTION

The **XTranslateCoordinates** function takes the src_x and src_y coordinates relative to the source window's origin and returns these coordinates to dest_x_return and dest_y_return relative to the destination window's origin. If **XTranslateCoordinates** returns zero, src_w and dest_w are on different screens, and dest_x_return and dest_y_return are zero. If the coordinates are contained in a mapped child of dest_w, that child is returned to child_return. Otherwise, child_return is set to **None**.

XTranslateCoordinates can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XUnmapEvent – UnmapNotify event structure

STRUCTURES

The structure for **UnmapNotify** events contains:

```
typedef struct {
        int type;                       /* UnmapNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window event;
        Window window;
        Bool from_configure;
} XUnmapEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The event member is set either to the unmapped window or to its parent, depending on whether **StructureNotify** or **SubstructureNotify** was selected. This is the window used by the X server to report the event. The window member is set to the window that was unmapped. The from_configure member is set to **True** if the event was generated as a result of a resizing of the window's parent when the window itself had a win_gravity of **UnmapGravity**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XVisibilityEvent(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XUnmapWindow, XUnmapSubwindows – unmap windows

SYNTAX

XUnmapWindow(*display*, *w*)
Display **display*;
Window *w*;

XUnmapSubwindows(*display*, *w*)
Display **display*;
Window *w*;

ARGUMENTS

display	Specifies the connection to the X server.
w	Specifies the window.

DESCRIPTION

The **XUnmapWindow** function unmaps the specified window and causes the X server to generate an **UnmapNotify** event. If the specified window is already unmapped, **XUnmapWindow** has no effect. Normal exposure processing on formerly obscured windows is performed. Any child window will no longer be visible until another map call is made on the parent. In other words, the subwindows are still mapped but are not visible until the parent is mapped. Unmapping a window will generate **Expose** events on windows that were formerly obscured by it.

XUnmapWindow can generate a **BadWindow** error.

The **XUnmapSubwindows** function unmaps all subwindows for the specified window in bottom-to-top stacking order. It causes the X server to generate an **UnmapNotify** event on each subwindow and **Expose** events on formerly obscured windows. Using this function is much more efficient than unmapping multiple windows one at a time because the server needs to perform much of the work only once, for all of the windows, rather than for each window.

XUnmapSubwindows can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XChangeWindowAttributes(3), XConfigureWindow(3), XCreateWindow(3), XDestroyWindow(3), XMapWindow(3) XRaiseWindow(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XVisibilityNotifyEvent – VisibilityNotify event structure

STRUCTURES

The structure for **VisibilityNotify** events contains:

```
typedef struct {
        int type;                       /* VisibiltyNotify */
        unsigned long serial;           /* # of last request processed by server */
        Bool send_event;                /* true if this came from a SendEvent request */
        Display *display;               /* Display the event was read from */
        Window window;
        int state;
} XVisibilityEvent;
```

When you receive this event, the structure members are set as follows.

The type member is set to the event type constant name that uniquely identifies it. For example, when the X server reports a **GraphicsExpose** event to a client application, it sends an **XGraphicsExposeEvent** structure with the type member set to **GraphicsExpose**. The display member is set to a pointer to the display the event was read on. The send_event member is set to **True** if the event came from a **SendEvent** protocol request. The serial member is set from the serial number reported in the protocol but expanded from the 16-bit least-significant bits to a full 32-bit value. The window member is set to the window that is most useful to toolkit dispatchers.

The window member is set to the window whose visibility state changes. The state member is set to the state of the window's visibility and can be **VisibilityUnobscured**, **VisibilityPartiallyObscured**, or **VisibilityFullyObscured**. The X server ignores all of a window's subwindows when determining the visibility state of the window and processes **VisibilityNotify** events according to the following:

- When the window changes state from partially obscured, fully obscured, or not viewable to viewable and completely unobscured, the X server generates the event with the state member of the **XVisibilityEvent** structure set to **VisibilityUnobscured**.
- When the window changes state from viewable and completely unobscured or not viewable to viewable and partially obscured, the X server generates the event with the state member of the **XVisibilityEvent** structure set to **VisibilityPartiallyObscured**.
- When the window changes state from viewable and completely unobscured, viewable and partially obscured, or not viewable to viewable and fully obscured, the X server generates the event with the state member of the **XVisibilityEvent** structure set to **VisibilityFullyObscured**.

SEE ALSO

XAnyEvent(3), XButtonEvent(3), XCreateWindowEvent(3), XCirculateEvent(3), XCirculateRequestEvent(3), XColormapEvent(3), XConfigureEvent(3), XConfigureRequestEvent(3), XCrossingEvent(3), XDestroyWindowEvent(3), XErrorEvent(3), XExposeEvent(3), XFocusChangeEvent(3), XGraphicsExposeEvent(3), XGravityEvent(3), XKeymapEvent(3), XMapEvent(3), XMapRequestEvent(3), XPropertyEvent(3), XReparentEvent(3), XResizeRequestEvent(3), XSelectionClearEvent(3), XSelectionEvent(3), XSelectionRequestEvent(3), XUnmapEvent(3),
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XWarpPointer – move pointer

SYNTAX

XWarpPointer(*display, src_w, dest_w, src_x, src_y, src_width, src_height, dest_x, dest_y*)
Display **display*;
Window *src_w, dest_w*;
int *src_x, src_y*;
unsigned int *src_width, src_height*;
int *dest_x, dest_y*;

ARGUMENTS

dest_w Specifies the destination window or **None**.

dest_x
dest_y Specify the x and y coordinates within the destination window.

display Specifies the connection to the X server.

src_x
src_y
src_width
src_height Specify a rectangle in the source window.

src_w Specifies the source window or **None**.

DESCRIPTION

If dest_w is **None**, **XWarpPointer** moves the pointer by the offsets (dest_x, dest_y) relative to the current position of the pointer. If dest_w is a window, **XWarpPointer** moves the pointer to the offsets (dest_x, dest_y) relative to the origin of dest_w. However, if src_w is a window, the move only takes place if the window src_w contains the pointer and if the pointer is contained in the specified rectangle of src_w.

The src_x and src_y coordinates are relative to the origin of src_w. If src_height is zero, it is replaced with the current height of src_w minus src_y. If src_width is zero, it is replaced with the current width of src_w minus src_x.

There is seldom any reason for calling this function. The pointer should normally be left to the user. If you do use this function, however, it generates events just as if the user had instantaneously moved the pointer from one position to another. Note that you cannot use **XWarpPointer** to move the pointer outside the confine_to window of an active pointer grab. An attempt to do so will only move the pointer as far as the closest edge of the confine_to window.

XWarpPointer can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a Window argument does not name a defined Window.

SEE ALSO

XSetInputFocus(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XauFileName, XauReadAuth, XauLockAuth, XauUnlockAuth, XauWriteAuth, XauGetAuthByAddr – X authority database routines

SYNOPSIS

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

typedef struct xauth {
        unsigned short      family;
        unsigned short      address_length;
        char                *address;
        unsigned short      number_length;
        char                *number;
        unsigned short      name_length;
        char                *name;
        unsigned short      data_length;
        char                *data;
} Xauth;

char *XauFileName ()

Xauth *XauReadAuth (auth_file)
        FILE *auth_file;

int XauWriteAuth (auth_file, auth)
        FILE *auth_file;
        Xauth *auth;

Xauth *XauGetAuthByAddr (family,
                         address_length, address,
                         number_length, number)
        unsigned short family;
        unsigned short address_length;
        char *address;
        unsigned short number_length;
        char *number;

int XauLockAuth (file_name, retries, timeout, dead)
        char *file_name;
        int retries;
        int timeout;
        long dead;

int XauUnlockAuth (file_name)
        char *file_name;

XauDisposeAuth (auth)
        Xauth *auth;
```

DESCRIPTION

XauFileName generates the default authorization file name by first checking the XAUTHROTIY environment variable if set, else it returns $HOME/.Xauthority. This name is statically allocated and should not be freed.

XauReadAuth reads the next entry from *auth_file*. The entry is **not** statically allocated and should be freed by calling *XauDisposeAuth*.

XuWriteAuth writes an authorization entry to *auth_file*. It returns 1 on success, 0 on failure.

XauGetAuthByAddr searches for an entry which matches the given network address/display number pair. The entry is **not** statically allocated and should be freed by calling *XauDisposeAuth*

XauLockAuth does the work necessary to synchronously update an authorization file. First it makes to file names, one with "-c" appended to *file_name*, the other with "-l" appended. If the "-c" file already exists and is more than *dead* seconds old, *XauLockAuth* removes it and the associated "-l" file. To prevent possible synchronization troubles with NFS, a *dead* value of zero forces the files to be removed. *XauLockAuth* makes *retries* attempts to create and link the file names, pausing *timeout* seconds between each attempt. *XauLockAuth* returns a collection of values depending on the results:

LOCK_ERROR	A system error occurred, either a file_name which is too long, or an unexpected failure from a system call. errno may prove useful.
LOCK_TIMEOUT	*retries* attempts failed
LOCK_SUCCESS	The lock succeeded.

XauUnlockAuth undoes the work of *XauLockAuth* by unlinking both the "-c" and "-l" file names.

XauDisposeAuth frees storage allocated to hold an authorization entry.

SEE ALSO

xauth(1), xdm(1)

AUTHOR

Keith Packard, MIT X Consortium

NAME

XrmGetResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource – retrieve database resources and search lists

SYNTAX

Bool XrmGetResource(*database, str_name, str_class, str_type_return, value_return*)
XrmDatabase *database*;
char **str_name*;
char **str_class*;
char ***str_type_return*;
XrmValue **value_return*;

Bool XrmQGetResource(*database, quark_name, quark_class, quark_type_return, value_return*)
XrmDatabase *database*;
XrmNameList *quark_name*;
XrmClassList *quark_class*;
XrmRepresentation **quark_type_return*;
XrmValue **value_return*;

typedef XrmHashTable *XrmSearchList;

Bool XrmQGetSearchList(*database, names, classes, list_return, list_length*)
XrmDatabase *database*;
XrmNameList *names*;
XrmClassList *classes*;
XrmSearchList *list_return*;
int *list_length*;

Bool XrmQGetSearchResource(*list, name, class, type_return, value_return*)
XrmSearchList *list*;
XrmName *name*;
XrmClass *class*;
XrmRepresentation **type_return*;
XrmValue **value_return*;

ARGUMENTS

class	Specifies the resource class.
classes	Specifies a list of resource classes.
database	Specifies the database that is to be used.
list	Specifies the search list returned by **XrmQGetSearchList**.
list_length	Specifies the number of entries (not the byte size) allocated for list_return.
list_return	Returns a search list for further use.
name	Specifies the resource name.
names	Specifies a list of resource names.
quark_class	Specifies the fully qualified class of the value being retrieved (as a quark).
quark_name	Specifies the fully qualified name of the value being retrieved (as a quark).
quark_type_return	Returns a pointer to the representation type of the destination (as a quark).
str_class	Specifies the fully qualified class of the value being retrieved (as a string).
str_name	Specifies the fully qualified name of the value being retrieved (as a string).
str_type_return	Returns a pointer to the representation type of the destination (as a string).

type_return Returns data representation type.

value_return Returns the value in the database.

DESCRIPTION

The **XrmGetResource** and **XrmQGetResource** functions retrieve a resource from the specified database. Both take a fully qualified name/class pair, a destination resource representation, and the address of a value (size/address pair). The value and returned type point into database memory; therefore, you must not modify the data.

The database only frees or overwrites entries on **XrmPutResource**, **XrmQPutResource**, or **XrmMergeDatabases**. A client that is not storing new values into the database or is not merging the database should be safe using the address passed back at any time until it exits. If a resource was found, both **XrmGetResource** and **XrmQGetResource** return **True**; otherwise, they return **False**.

The **XrmQGetSearchList** function takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order and uses the same algorithm as **XrmGetResource** for determining precedence. If list_return was large enough for the search list, **XrmQGetSearchList** returns **True**; otherwise, it returns **False**.

The size of the search list that the caller must allocate is dependent upon the number of levels and wildcards in the resource specifiers that are stored in the database. The worst case length is 3^n, where n is the number of name or class components in names or classes.

When using **XrmQGetSearchList** followed by multiple probes for resources with a common name and class prefix, only the common prefix should be specified in the name and class list to **XrmQGetSearchList**.

The **XrmQGetSearchResource** function searches the specified database levels for the resource that is fully identified by the specified name and class. The search stops with the first match. **XrmQGetSearchResource** returns **True** if the resource was found; otherwise, it returns **False**.

A call to **XrmQGetSearchList** with a name and class list containing all but the last component of a resource name followed by a call to **XrmQGetSearchResource** with the last component name and class returns the same database entry as **XrmGetResource** and **XrmQGetResource** with the fully qualified name and class.

SEE ALSO

XrmInitialize(3), XrmMergeDatabases(3), XrmPutResource(3), XrmUniqueQuark(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XrmInitialize, XrmParseCommand, XrmValue, XrmOptionKind, XrmOptionDescRec – initialize the Resource Manager, Resource Manager structures, and parse the command line

SYNTAX

void XrmInitialize( );

void XrmParseCommand(*database*, *table*, *table_count*, *name*, *argc_in_out*, *argv_in_out*,)
XrmDatabase **database*;
XrmOptionDescList *table*;
int *table_count*;
char **name*;
int **argc_in_out*;
char ***argv_in_out*;

ARGUMENTS

argc_in_out	Specifies the number of arguments and returns the number of remaining arguments.
argv_in_out	Specifies a pointer to the command line arguments and returns the remaining arguments.
database	Specifies the resource database.
name	Specifies the application name.
table	Specifies the table of command line arguments to be parsed.
table_count	Specifies the number of entries in the table.

DESCRIPTION

The **XrmInitialize** function initialize the resource manager.

The **XrmParseCommand** function parses an (argc, argv) pair according to the specified option table, loads recognized options into the specified database with type "String," and modifies the (argc, argv) pair to remove all recognized options.

The specified table is used to parse the command line. Recognized entries in the table are removed from argv, and entries are made in the specified resource database. The table entries contain information on the option string, the option name, the style of option, and a value to provide if the option kind is **XrmoptionNoArg**. The argc argument specifies the number of arguments in argv and is set to the remaining number of arguments that were not parsed. The name argument should be the name of your application for use in building the database entry. The name argument is prefixed to the resourceName in the option table before storing the specification. No separating (binding) character is inserted. The table must contain either a period (.) or an asterisk (*) as the first character in each resourceName entry. To specify a more completely qualified resource name, the resourceName entry can contain multiple components.

STRUCTURES

The **XrmValue**, **XrmOptionKind**, and **XrmOptionDescRec** structures contain:

```
typedef struct {
        unsigned int size;
        caddr_t addr;
} XrmValue, *XrmValuePtr;

typedef enum {
        XrmoptionNoArg,         /* Value is specified in XrmOptionDescRec.value */
        XrmoptionIsArg,         /* Value is the option string itself */
        XrmoptionStickyArg,     /* Value is characters immediately following option */
        XrmoptionSepArg,        /* Value is next argument in argv */
        XrmoptionResArg,        /* Resource and value in next argument in argv */
```

```
        XrmoptionSkipArg,           /* Ignore this option and the next argument in argv */
        XrmoptionSkipLine,          /* Ignore this option and the rest of argv */
        XrmoptionSkipNArgs          /* Ignore this option and the next
                                       XrmOptionDescRec.value arguments in argv */
} XrmOptionKind;

typedef struct {
        char *option;               /* Option specification string in argv    */
        char *specifier;            /* Binding and resource name (sans application name)  */
        XrmOptionKind argKind;      /* Which style of option it is    */
        caddr_t value;              /* Value to provide if XrmoptionNoArg or
                                       XrmoptionSkipNArgs   */
} XrmOptionDescRec, *XrmOptionDescList;
```

SEE ALSO

XrmGetResource(3), XrmMergeDatabases(3), XrmPutResource(3), XrmUniqueQuark(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XrmMergeDatabases, XrmGetFileDatabase, XrmPutFileDatabase, XrmGetStringDatabase, XrmDestroyDatabase – manipulate resource databases

SYNTAX

void XrmMergeDatabases(*source_db*, *target_db*)
XrmDatabase *source_db*, **target_db*;

XrmDatabase XrmGetFileDatabase(*filename*)
char **filename*;

void XrmPutFileDatabase(*database*, *stored_db*)
XrmDatabase *database*;
char **stored_db*;

XrmDatabase XrmGetStringDatabase(*data*)
char **data*;

void XrmDestroyDatabase(*database*)
XrmDatabase *database*;

ARGUMENTS

data	Specifies the database contents using a string.
database	Specifies the database that is to be used.
filename	Specifies the resource database file name.
source_db	Specifies the resource database that is to be merged into the target database.
stored_db	Specifies the file name for the stored database.
target_db	Specifies a pointer to the resource database into which the source database is to be merged.

DESCRIPTION

The **XrmMergeDatabases** function merges the contents of one database into another. It may overwrite entries in the destination database. This function is used to combine databases (for example, an application specific database of defaults and a database of user preferences). The merge is destructive; that is, the source database is destroyed.

The **XrmGetFileDatabase** function opens the specified file, creates a new resource database, and loads it with the specifications read in from the specified file. The specified file must contain lines in the format accepted by **XrmPutLineResource**. If it cannot open the specified file, **XrmGetFileDatabase** returns NULL.

The **XrmPutFileDatabase** function stores a copy of the specified database in the specified file. The file is an ASCII text file that contains lines in the format that is accepted by **XrmPutLineResource**.

The **XrmGetStringDatabase** function creates a new database and stores the resources specified in the specified null-terminated string. **XrmGetStringDatabase** is similar to **XrmGetFileDatabase** except that it reads the information out of a string instead of out of a file. Each line is separated by a new-line character in the format accepted by **XrmPutLineResource**.

If database is NULL, **XrmDestroyDatabase** returns immediately.

SEE ALSO

XrmGetResource(3), XrmInitialize(3), XrmPutResource(3), XrmUniqueQuark(3)
Xlib – C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XrmPutResource, XrmQPutResource, XrmPutStringResource, XrmQPutStringResource, XrmPutLineResource – store database resources

SYNTAX

void XrmPutResource(*database, specifier, type, value*)
XrmDatabase **database*;
char **specifier*;
char **type*;
XrmValue **value*;

void XrmQPutResource(*database, bindings, quarks, type, value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
XrmRepresentation *type*;
XrmValue **value*;

void XrmPutStringResource(*database, specifier, value*)
XrmDatabase **database*;
char **specifier*;
char **value*;

void XrmQPutStringResource(*database, bindings, quarks, value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
char **value*;

void XrmPutLineResource(*database, line*)
XrmDatabase **database*;
char **line*;

ARGUMENTS

bindings	Specifies a list of bindings.
database	Specifies the resource database.
line	Specifies the resource name and value pair as a single string in the valid ResourceLine format (see section 10.11). A single colon (:) separates the name from the value. Note that comment lines are not stored.
quarks	Specifies the complete or partial name or the class list of the resource.
specifier	Specifies a complete or partial specification of the resource.
type	Specifies the type of the resource.
value	Specifies the value of the resource, which is specified as a string.

DESCRIPTION

If database contains NULL, **XrmPutResource** creates a new database and returns a pointer to it. **XrmPutResource** is a convenience function that calls **XrmStringToBindingQuarkList** followed by:

XrmQPutResource(database, bindings, quarks, XrmStringToQuark(type), value)

If database contains NULL, **XrmQPutResource** creates a new database and returns a pointer to it.

If database contains NULL, **XrmPutStringResource** creates a new database and returns a pointer to it. **XrmPutStringResource** adds a resource with the specified value to the specified database. **XrmPutStringResource** is a convenience function that first calls

XrmStringToBindingQuarkList on the specifier and then calls **XrmQPutResource**, using a "String" representation type.

If database contains NULL, **XrmQPutStringResource** creates a new database and returns a pointer to it. **XrmQPutStringResource** is a convenience routine that constructs an **XrmValue** for the value string (by calling **strlen** to compute the size) and then calls **XrmQPutResource**, using a "String" representation type.

If database contains NULL, **XrmPutLineResource** creates a new database and returns a pointer to it. **XrmPutLineResource** adds a single resource entry to the specified database. Any white space before or after the name or colon in the line argument is ignored. The value is terminated by a new-line or a NULL character. To allow values to contain embedded new-line characters, a "\n" is recognized and replaced by a new-line character. For example, line might have the value "xterm*background:green\n". Null-terminated strings without a new line are also permitted.

To allow values to contain arbitrary octets, the 4-character sequence *\nnn*, where n is a digit in the range of "0"–"7", is recognized and replaced with a single byte that contains this sequence interpreted as an octal number. For example, a value containing a NULL byte can be stored by specifying "\000" in the string.

SEE ALSO

XrmGetResource(3), XrmInitialize(3), XrmMergeDatabases(3), XrmUniqueQuark(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XrmUniqueQuark, XrmStringToQuark, XrmQuarkToString, XrmStringToQuarkList, XrmStringToBindingQuarkList – manipulate resource quarks

SYNTAX

XrmQuark XrmUniqueQuark( )

#define XrmStringToName(string) XrmStringToQuark(string) #define XrmStringToClass(string) XrmStringToQuark(string) #define XrmStringToRepresentation(string) XrmStringToQuark(string)

XrmQuark XrmStringToQuark(*string*)
char **string*;

#define XrmNameToString(name) XrmQuarkToString(name) #define XrmClassToString(class) XrmQuarkToString(class) #define XrmRepresentationToString(type) XrmQuarkToString(type)

char *XrmQuarkToString(*quark*)
XrmQuark *quark*;

#define XrmStringToNameList(str, name) XrmStringToQuarkList((str), (name)) #define XrmStringToClassList(str,class) XrmStringToQuarkList((str), (class))

void XrmStringToQuarkList(*string, quarks_return*)
char **string*;
XrmQuarkList *quarks_return*;

XrmStringToBindingQuarkList(*string, bindings_return, quarks_return*)
char **string*;
XrmBindingList *bindings_return*;
XrmQuarkList *quarks_return*;

ARGUMENTS

bindings_return	Returns the binding list.
quark	Specifies the quark for which the equivalent string is desired.
quarks_return	Returns the list of quarks.
string	Specifies the string for which a quark is to be allocated.

DESCRIPTION

The **XrmUniqueQuark** function allocates a quark that is guaranteed not to represent any string that is known to the resource manager.

These functions can be used to convert to and from quark representations. The string pointed to by the return value must not be modified or freed. If no string exists for that quark, **XrmQuarkToString** returns NULL.

The **XrmQuarkToString** function converts the specified resource quark representation back to a string.

The **XrmStringToQuarkList** function converts the null-terminated string (generally a fully qualified name) to a list of quarks. Note that the string must be in the valid ResourceName format (see section 10.11). The components of the string are separated by a period or asterisk character.

A binding list is a list of type **XrmBindingList** and indicates if components of name or class lists are bound tightly or loosely (that is, if wildcarding of intermediate components is specified).

typedef enum {XrmBindTightly, XrmBindLoosely} XrmBinding, *XrmBindingList;

XrmBindTightly indicates that a period separates the components, and **XrmBindLoosely** indicates that an asterisk separates the components.

The **XrmStringToBindingQuarkList** function converts the specified string to a binding list and a quark list. Component names in the list are separated by a period or an asterisk character. If the string does not start with period or asterisk, a period is assumed. For example, "*a.b*c" becomes:

quarks	a	b	c
bindings	loose	tight	loose

SEE ALSO

XrmGetResource(3), XrmInitialize(3), XrmMergeDatabases(3), XrmPutResource(3)
Xlib - C Language X Interface

NOTES

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAddCallback, XtAddCallbacks, XtRemoveCallback, XtRemoveCallbacks, XtRemoveAllCallbacks – add and remove callback procedures

SYNTAX

void XtAddCallback(*w, callback_name, callback, client_data*)
Widget *w*;
String *callback_name*;
XtCallbackProc *callback*;
XtPointer *client_data*;

void XtAddCallbacks(*w, callback_name, callbacks*)
Widget *w*;
String *callback_name*;
XtCallbackList *callbacks*;

void XtRemoveCallback(*w, callback_name, callback, client_data*)
Widget *w*;
String *callback_name*;
XtCallbackProc *callback*;
XtPointer *client_data*;

void XtRemoveCallbacks(*w, callback_name, callbacks*)
Widget *w*;
String *callback_name*;
XtCallbackList *callbacks*;

void XtRemoveAllCallbacks(*w, callback_name*)
Widget *w*;
String *callback_name*;

ARGUMENTS

callback	Specifies the callback procedure.
callbacks	Specifies the null-terminated list of callback procedures and corresponding client data.
callback_name	Specifies the callback list to which the procedure is to be appended or deleted.
client_data	Specifies the argument that is to be passed to the specified procedure when it is invoked by XtCallbacks or NULL, or the client data to match on the registered callback procedures.
w	Specifies the widget.

DESCRIPTION

The **XtAddCallback** function adds the specified callback procedure to the specified widget's callback list.

The **XtAddCallbacks** add the specified list of callbacks to the specified widget's callback list.

The **XtRemoveCallback** function removes a callback only if both the procedure and the client data match.

The **XtRemoveCallbacks** function removes the specified callback procedures from the specified widget's callback list.

The **XtRemoveAllCallbacks** function removes all the callback procedures from the specified widget's callback list.

SEE ALSO

XtCallCallbacks(3)
X Toolkit Intrinsics – C Language Interface

Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAddEventHandler, XtAddRawEventHandler, XtRemoveEventHandler XtRemoveRawEventHandler – add and remove event handlers

SYNTAX

void XtAddEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
XtPointer *client_data*;

void XtAddRawEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
XtPointer *client_data*;

void XtRemoveEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
XtPointer *client_data*;

void XtRemoveRawEventHandler(*w, event_mask, nonmaskable, proc, client_data*)
Widget *w*;
EventMask *event_mask*;
Boolean *nonmaskable*;
XtEventHandler *proc*;
XtPointer *client_data*;

ARGUMENTS

client_data	Specifies additional data to be passed to the client's event handler.
event_mask	Specifies the event mask for which to call or unregister this procedure.
nonmaskable	Specifies a Boolean value that indicates whether this procedure should be called or removed on the nonmaskable events (**GraphicsExpose**, **NoExpose**, **SelectionClear**, **SelectionRequest**, **SelectionNotify**, **ClientMessage**, and **MappingNotify**).
proc	Specifies the procedure that is to be added or removed.
w	Specifies the widget for which this event handler is being registered.

DESCRIPTION

The **XtAddEventHandler** function registers a procedure with the dispatch mechanism that is to be called when an event that matches the mask occurs on the specified widget. If the procedure is already registered with the same client_data, the specified mask is ORed into the existing mask. If the widget is realized, **XtAddEventHandler** calls **XSelectInput**, if necessary.

The **XtAddRawEventHandler** function is similar to **XtAddEventHandler** except that it does not affect the widget's mask and never causes an **XSelectInput** for its events. Note that the widget might already have those mask bits set because of other nonraw event handlers registered on it.

The **XtAddRawEventHandler** function is similar to **XtAddEventHandler** except that it does not affect the widget's mask and never causes an **XSelectInput** for its events. Note that the widget might already have those mask bits set because of other nonraw event handlers registered

on it.

The **XtRemoveRawEventHandler** function stops the specified procedure from receiving the specified events. Because the procedure is a raw event handler, this does not affect the widget's mask and never causes a call on **XSelectInput**.

SEE ALSO

XtAppNextEvent(3), XtBuildEventMask(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAddExposureToRegion – merge exposure events into a region

SYNTAX

void XtAddExposureToRegion(*event, region*)
XEvent **event*;
Region *region*;

ARGUMENTS

event	Specifies a pointer to the **Expose** or **GraphicsExpose** event.
region	Specifies the region object (as defined in <**X11/Xutil.h**>).

DESCRIPTION

The **XtAddExposureToRegion** function computes the union of the rectangle defined by the exposure event and the specified region. Then, it stores the results back in region. If the event argument is not an **Expose** or **GraphicsExpose** event, **XtAddExposureToRegion** returns without an error and without modifying region.

This function is used by the exposure compression mechanism (see Section 7.9.3).

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAddGrab, XtRemoveGrab – redirect user input to a modal widget

SYNTAX

void XtAddGrab(*w*, *exclusive*, *spring_loaded*)
Widget *w*;
Boolean *exclusive*;
Boolean *spring_loaded*;

void XtRemoveGrab(*w*)
Widget *w*;

ARGUMENTS

exclusive	Specifies whether user events should be dispatched exclusively to this widget or also to previous widgets in the cascade.
spring_loaded	Specifies whether this widget was popped up because the user pressed a pointer button.
w	Specifies the widget to add to or remove from the modal cascade.

DESCRIPTION

The **XtAddGrab** function appends the widget (and associated parameters) to the modal cascade and checks that exclusive is **True** if spring_loaded is **True**. If these are not **True**, **XtAddGrab** generates an error.

The modal cascade is used by **XtDispatchEvent** when it tries to dispatch a user event. When at least one modal widget is in the widget cascade, **XtDispatchEvent** first determines if the event should be delivered. It starts at the most recent cascade entry and follows the cascade up to and including the most recent cascade entry added with the exclusive parameter **True**.

This subset of the modal cascade along with all descendants of these widgets comprise the active subset. User events that occur outside the widgets in this subset are ignored or remapped. Modal menus with submenus generally add a submenu widget to the cascade with exclusive **False**. Modal dialog boxes that need to restrict user input to the most deeply nested dialog box add a subdialog widget to the cascade with exclusive **True**. User events that occur within the active subset are delivered to the appropriate widget, which is usually a child or further descendant of the modal widget.

Regardless of where on the screen they occur, remap events are always delivered to the most recent widget in the active subset of the cascade that has spring_loaded **True**, if any such widget exists.

The **XtRemoveGrab** function removes widgets from the modal cascade starting at the most recent widget up to and including the specified widget. It issues an error if the specified widget is not on the modal cascade.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppAddActions – register an action table

SYNTAX

void XtAppAddActions(*app_context*, *actions*, *num_actions*)
XtAppContext *app_context*;
XtActionList *actions*;
Cardinal *num_actions*;

ARGUMENTS

app_context	Specifies the application context.
actions	Specifies the action table to register.
num_args	Specifies the number of entries in this action table.

DESCRIPTION

The **XtAppAddActions** function adds the specified action table and registers it with the translation manager.

SEE ALSO

XtParseTranslationTable(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppAddConverter - register resource converter

SYNTAX

void XtAppAddConverter(*app_context, from_type, to_type, converter, convert_args, num_args*)
XtAppContext *app_context*;
String *from_type*;
String *to_type*;
XtConverter *converter*;
XtConvertArgList *convert_args*;
Cardinal *num_args*;

ARGUMENTS

app_context	Specifies the application context.
converter	Specifies the type converter procedure.
convert_args	Specifies how to compute the additional arguments to the converter or NULL.
from_type	Specifies the source type.
num_args	Specifies the number of additional arguments to the converter or zero.
to_type	Specifies the destination type.

DESCRIPTION

The **XtAppAddConverter** registers a the specified resource converter.

SEE ALSO

XtConvert(3), XtStringConversionWarning(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppAddInput, XtRemoveInput – register and remove an input source

SYNTAX

XtInputId XtAppAddInput(*app_context, source, condition, proc, client_data*)
XtAppContext *app_context*;
int *source*;
XtPointer *condition*;
XtInputCallbackProc *proc*;
XtPointer *client_data*;

void XtRemoveInput(*id*)
XtInputId *id*;

ARGUMENTS

app_context	Specifies the application context that identifies the application.
client_data	Specifies the argument that is to be passed to the specified procedure when input is available.
condition	Specifies the mask that indicates a read, write, or exception condition or some operating system dependent condition.
id	Specifies the ID returned from the corresponding **XtAppAddInput** call.
proc	Specifies the procedure that is to be called when input is available.
source	Specifies the source file descriptor on a UNIX-based system or other operating system dependent device specification.

DESCRIPTION

The **XtAppAddInput** function registers with the Intrinsics read routine a new source of events, which is usually file input but can also be file output. Note that file should be loosely interpreted to mean any sink or source of data. **XtAppAddInput** also specifies the conditions under which the source can generate events. When input is pending on this source, the callback procedure is called.

The legal values for the condition argument are operating-system dependent. On a UNIX-based system, the condition is some union of **XtInputReadMask**, **XtInputWriteMask**, and **XtInputExceptMask**. The **XtRemoveInput** function causes the Intrinsics read routine to stop watching for input from the input source.

SEE ALSO

XtAppAddTimeOut(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppAddTimeOut, XtRemoveTimeOut – register and remove timeouts

SYNTAX

XtIntervalId XtAppAddTimeOut(*app_context, interval, proc, client_data*)
XtAppContext *app_context*;
unsigned long *interval*;
XtTimerCallbackProc *proc*;
XtPointer *client_data*;

void XtRemoveTimeOut(*timer*)
XtIntervalId *timer*;

ARGUMENTS

app_context	Specifies the application context for which the timer is to be set.
client_data	Specifies the argument that is to be passed to the specified procedure when input is available.
interval	Specifies the time interval in milliseconds.
proc	Specifies the procedure that is to be called when time expires.
timer	Specifies the ID for the timeout request to be destroyed.

DESCRIPTION

The **XtAppAddTimeOut** function creates a timeout and returns an identifier for it. The timeout value is set to interval. The callback procedure is called when the time interval elapses, and then the timeout is removed.

The **XtRemoveTimeOut** function removes the timeout. Note that timeouts are automatically removed once they trigger.

SEE ALSO

XtAppAddInput(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppAddWorkProc, XtRemoveWorkProc – Add and remove background processing procedures

SYNTAX

XtWorkProcId XtAppAddWorkProc(*app_context*, *proc*, *client_data*)
XtAppContext *app_context*;
XtWorkProc *proc*;
XtPointer *client_data*;

void XtRemoveWorkProc(*id*)
XtWorkProcId *id*;

ARGUMENTS

app_context	Specifies the application context that identifies the application.
client_data	Specifies the argument that is to be passed to the specified procedure when it is called.
proc	Specifies the procedure that is to be called when time expires.
id	Specifies which work procedure to remove.

DESCRIPTION

The **XtAppAddWorkProc** function adds the specified work procedure for the application identified by app_context.

The **XtRemoveWorkProc** function explicitly removes the specified background work procedure.

SEE ALSO

XtAppNextEvent(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppCreateShell – create top-level widget instance

SYNTAX

Widget XtAppCreateShell(*application_name, application_class, widget_class, display, args, num_args*)

String *application_name*;
String *application_class*;
WidgetClass *widget_class*;
Display **display*;
ArgList *args*;
Cardinal *num_args*;

ARGUMENTS

application_class
Specifies the class name of this application.

application_name
Specifies the name of the application instance.

args Specifies the argument list in which to set in the WM_COMMAND property.

display Specifies the display from which to get the resources.

num_args Specifies the number of arguments in the argument list.

widget_class Specifies the widget class that the application top-level widget should be.

DESCRIPTION

The **XtAppCreateShell** function saves the specified application name and application class for qualifying all widget resource specifiers. The application name and application class are used as the left-most components in all widget resource names for this application. **XtAppCreateShell** should be used to create a new logical application within a program or to create a shell on another display. In the first case, it allows the specification of a new root in the resource hierarchy. In the second case, it uses the resource database associated with the other display.

Note that the widget returned by **XtAppCreateShell** has the WM_COMMAND property set for session managers (see Chapter 4).

SEE ALSO

XtCreateWidget(3)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppError, XtAppSetErrorHandler, XtAppSetWarningHandler, XtAppWarning – low-level error handlers

SYNTAX

void XtAppError(*app_context*, *message*)
XtAppContext *app_context*;
String *message*;

void XtAppSetErrorHandler(*app_context*, *handler*)
XtAppContext *app_context*;
XtErrorHandler *handler*;

void XtAppSetWarningHandler(*app_context*, *handler*)
XtAppContext *app_context*;
XtErrorHandler *handler*;

void XtAppWarning(*app_context*, *message*)
XtAppContext *app_context*;
String *message*;

ARGUMENTS

app_context	Specifies the application context.
message	Specifies the nonfatal error message that is to be reported.
handler	Specifies the new fatal error procedure, which should not return, or the nonfatal error procedure, which usually returns.
message	Specifies the message that is to be reported.

DESCRIPTION

The **XtAppError** function calls the installed error procedure and passes the specified message.

The **XtAppSetErrorHandler** function registers the specified procedure, which is called when a fatal error condition occurs.

The **XtAppSetWarningHandler** registers the specified procedure, which is called when a nonfatal error condition occurs.

The **XtAppWarning** function calls the installed nonfatal error procedure and passes the specified message.

SEE ALSO

XtAppGetErrorDatabase(3), XtAppErrorMsg(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppErrorMsg, XtAppSetErrorMsgHandler, XtAppSetWarningMsgHandler, XtAppWarningMsg – high-level error handlers

SYNTAX

void XtAppErrorMsg(*app_context, name, type, class, default, params, num_params*)
XtAppContext *app_context*;
String *name*;
String *type*;
String *class*;
String *default*;
String **params*;
Cardinal **num_params*;

void XtAppSetErrorMsgHandler(*app_context, msg_handler*)
XtAppContext *app_context*;
XtErrorMsgHandler *msg_handler*;

void XtAppSetWarningMsgHandler(*app_context, msg_handler*)
XtAppContext *app_context*;
XtErrorMsgHandler *msg_handler*;

void XtAppWarningMsg(*app_context, name, type, class, default, params, num_params*)
XtAppContext *app_context*;
String *name*;
String *type*;
String *class*;
String *default*;
String **params*;
Cardinal **num_params*;

ARGUMENTS

app_context	Specifies the application context.
class	Specifies the resource class.
default	Specifies the default message to use.
name	Specifies the general kind of error.
type	Specifies the detailed name of the error.
msg_handler	Specifies the new fatal error procedure, which should not return or the nonfatal error procedure, which usually returns.
num_params	Specifies the number of values in the parameter list.
params	Specifies a pointer to a list of values to be stored in the message.

DESCRIPTION

The **XtAppErrorMsg** function calls the high-level error handler and passes the specified information.

The **XtAppSetErrorMsgHandler** function registers the specified procedure, which is called when a fatal error occurs.

The **XtAppSetWarningMsgHandler** function registers the specified procedure, which is called when a nonfatal error condition occurs.

The **XtAppWarningMsg** function calls the high-level error handler and passes the specified information.

SEE ALSO

XtAppGetErrorDatabase(3), XtAppError(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppGetErrorDatabase, XtAppGetErrorDatabaseText – obtain error database

SYNTAX

XrmDatabase *XtAppGetErrorDatabase(*app_context*)
XtAppContext *app_context*;

void XtAppGetErrorDatabaseText(*app_context, name, type, class, default, buffer_return, nbytes, database*)
XtAppContext *app_context*;
char **name*, **type*, **class*;
char **default*;
char **buffer_return*;
int *nbytes*;
XrmDatabase *database*;

ARGUMENTS

app_context	Specifies the application context.
buffer_return	Specifies the buffer into which the error message is to be returned.
class	Specifies the resource class of the error message.
database	Specifies the name of the alternative database that is to be used or NULL if the application's database is to be used.
default	Specifies the default message to use.
name	
type	Specifies the name and type that are concatenated to form the resource name of the error message.
nbytes	Specifies the size of the buffer in bytes.

DESCRIPTION

The **XtAppGetErrorDatabase** function returns the address of the error database. The Intrinsics do a lazy binding of the error database and do not merge in the database file until the first call to **XtAppGetErrorDatbaseText**.

The **XtAppGetErrorDatabaseText** returns the appropriate message from the error database or returns the specified default message if one is not found in the error database.

SEE ALSO

XtAppError(3), XtAppErrorMsg(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppGetSelectionTimeout, XtAppSetSelectionTimeout– set and obtain selection timeout values

SYNTAX

unsigned long XtAppGetSelectionTimeout(*app_context*)
XtAppContext *app_context*;

void XtAppSetSelectionTimeout(*app_context*, *timeout*)
XtAppContext *app_context*;
unsigned long *timeout*;

ARGUMENTS

app_context	Specifies the application context.
timeout	Specifies the selection timeout in milliseconds.

DESCRIPTION

The **XtAppGetSelectionTimeout** function returns the current selection timeout value, in milliseconds. The selection timeout is the time within which the two communicating applications must respond to one another. The initial timeout value is set by the **selectionTimeout** application resource, or, if **selectionTimeout** is not specified, it defaults to five seconds.

The **XtAppSetSelectionTimeout** function sets the Intrinsics's selection timeout mechanism. Note that most applications should not set the selection timeout.

SEE ALSO

XtOwnSelection(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtAppNextEvent, XtAppPending, XtAppPeekEvent, XtAppProcessEvent, XtDispatchEvent, XtAppMainLoop – query and process events and input

SYNTAX

void XtAppNextEvent(*app_context, event_return*)
XtAppContext *app_context*;
XEvent **event_return*;

Boolean XtAppPeekEvent(*app_context, event_return*)
XtAppContext *app_context*;
XEvent **event_return*;

XtInputMask XtAppPending(*app_context*)
XtAppContext *app_context*;

void XtAppProcessEvent(*app_context, mask*)
XtAppContext *app_context*;
XtInputMask *mask*;

Boolean XtDispatchEvent(*event*)
XEvent **event*;

void XtAppMainLoop(*app_context*)
XtAppContext *app_context*;

ARGUMENTS

app_context	Specifies the application context that identifies the application .
event	Specifies a pointer to the event structure that is to be dispatched to the appropriate event handler.
event_return	Returns the event information to the specified event structure.
mask	Specifies what types of events to process. The mask is the bitwise inclusive OR of any combination of **XtIMXEvent**, **XtIMTimer**, and **XtIMAlternateInput**. As a convenience, the X Toolkit defines the symbolic name **XtIMAll** to be the bitwise inclusive OR of all event types.

DESCRIPTION

If no input is on the X input queue, **XtAppNextEvent** flushes the X output buffer and waits for an event while looking at the other input sources and timeout values and calling any callback procedures triggered by them. This wait time can be used for background processing (see Section 7.8).

If there is an event in the queue, **XtAppPeekEvent** fills in the event and returns a nonzero value. If no X input is on the queue, **XtAppPeekEvent** flushes the output buffer and blocks until input is available (possibly calling some timeout callbacks in the process). If the input is an event, **XtAppPeekEvent** fills in the event and returns a nonzero value. Otherwise, the input is for an alternate input source, and **XtAppPeekEvent** returns zero.

The **XtAppPending** function returns a nonzero value if there are events pending from the X server, timer pending, or other input sources pending. The value returned is a bit mask that is the OR of **XtIMXEvent**, **XtIMTimer**, and **XtIMAlternateInput** (see **XtAppProcessEvent**). If there are no events pending, **XtAppPending** flushes the output buffer and returns zero.

The **XtAppProcessEvent** function processes one timer, alternate input, or X event. If there is nothing of the appropriate type to process, **XtAppProcessEvent** blocks until there is. If there is more than one type of thing available to process, it is undefined which will get processed. Usually, this procedure is not called by client applications (see **XtAppMainLoop**). **XtAppProcessEvent** processes timer events by calling any appropriate timer callbacks, alternate input by

calling any appropriate alternate input callbacks, and X events by calling **XtDispatchEvent**.

When an X event is received, it is passed to **XtDispatchEvent**, which calls the appropriate event handlers and passes them the widget, the event, and client-specific data registered with each procedure. If there are no handlers for that event registered, the event is ignored and the dispatcher simply returns. The order in which the handlers are called is undefined.

The **XtDispatchEvent** function sends those events to the event handler functions that have been previously registered with the dispatch routine. **XtDispatchEvent** returns **True** if it dispatched the event to some handler and **False** if it found no handler to dispatch the event to. The most common use of **XtDispatchEvent** is to dispatch events acquired with the **XtAppNextEvent** procedure. However, it also can be used to dispatch user-constructed events. **XtDispatchEvent** also is responsible for implementing the grab semantics for **XtAddGrab**.

The **XtAppMainLoop** function first reads the next incoming X event by calling **XtAppNextEvent** and then it dispatches the event to the appropriate registered procedure by calling **XtDispatchEvent**. This constitutes the main loop of X Toolkit applications, and, as such, it does not return. Applications are expected to exit in response to some user action. There is nothing special about **XtAppMainLoop**; it is simply an infinite loop that calls **XtAppNextEvent** and then **XtDispatchEvent**.

Applications can provide their own version of this loop, which tests some global termination flag or tests that the number of top-level widgets is larger than zero before circling back to the call to **XtAppNextEvent**.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtBuildEventMask – retrieve a widget's event mask

SYNTAX

EventMask XtBuildEventMask(*w*)
Widget *w*;

ARGUMENTS

w Specifies the widget.

DESCRIPTION

The **XtBuildEventMask** function returns the event mask representing the logical OR of all event masks for event handlers registered on the widget with **XtAddEventHandler** and all event translations, including accelerators, installed on the widget. This is the same event mask stored into the **XSetWindowAttributes** structure by **XtRealizeWidget** and sent to the server when event handlers and translations are installed or removed on the realized widget.

SEE ALSO

XtAddEventHandler(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtCallAcceptFocus - calla widget's accept_focus procedure

SYNTAX

Boolean XtCallAcceptFocus(*w*, *time*)
Widget *w*;
Time **time*;

ARGUMENTS

time	Specifies the X time of the event that is causing the accept focus.
w	Specifies the widget.

DESCRIPTION

The **XtCallAcceptFocus** function calls the specified widget's accept_focus procedure, passing it the specified widget and time, and returns what the accept_focus procedure returns. If accept_focus is NULL, **XtCallAcceptFocus** returns **False**.

SEE ALSO

XtSetKeyboardFocus(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtCallCallbacks, XtHasCallbacks - process callbacks

SYNTAX

void XtCallCallbacks(*w, callback_name, call_data*)
Widget *w*;
String *callback_name*;
XtPointer *call_data*;

typedef enum {XtCallbackNoList, XtCallbackHasNone, XtCallbackHasSome} XtCallbackStatus;

XtCallbackStatus XtHasCallbacks(*w, callback_name*)
Widget *w*;
String *callback_name*;

ARGUMENTS

callback_name	Specifies the callback list to be executed or checked.
call_data	Specifies a callback-list specific data value to pass to each of the callback procedure in the list.
w	Specifies the widget.

DESCRIPTION

The **XtCallCallbacks** function calls each procedure that is registered in the specified widget's callback list.

The **XtHasCallbacks** function first checks to see if the widget has a callback list identified by callback_name. If the callback list does not exist, **XtHasCallbacks** returns **XtCallbackNoList**. If the callback list exists but is empty, it returns **XtCallbackHasNone**. If the callback list exists and has at least one callback registered, it returns **XtCallbackHasSome**.

SEE ALSO

XtAddCallback(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtClass, XtSuperClass, XtIsSubclass, XtCheckSubclass, XtIsComposite, XtIsManaged - obtain and verify a widget's class

SYNTAX

WidgetClass XtClass(*w*)
Widget *w*;

WidgetClass XtSuperclass(*w*)
Widget *w*;

Boolean XtIsSubclass(*w*, *widget_class*)
Widget *w*;
WidgetClass *widget_class*;

void XtCheckSubclass(*w*, *widget_class*, *message*)
Widget *w*;
WidgetClass *widget_class*;
String *message*;

Boolean XtIsComposite(*w*)
Widget *w*;

Boolean XtIsManaged(*w*)
Widget *w*;

ARGUMENTS

w	Specifies the widget.
widget_class	Specifies the widget class that the application top-level widget should be.
message	Specifies the message that is to be used.

DESCRIPTION

The **XtClass** function returns a pointer to the widget's class structure.

The **XtSuperclass** function returns a pointer to the widget's superclass class structure.

The **XtIsSubclass** function returns **True** if the class of the specified widget is equal to or is a subclass of the specified widget class. The specified widget can be any number of subclasses down the chain and need not be an immediate subclass of the specified widget class. Composite widgets that need to restrict the class of the items they contain can use **XtIsSubclass** to find out if a widget belongs to the desired class of objects.

The **XtCheckSubclass** macro determines if the class of the specified widget is equal to or is a subclass of the specified widget class. The widget can be any number of subclasses down the chain and need not be an immediate subclass of the specified widget class. If the specified widget is not a subclass, **XtCheckSubclass** constructs an error message from the supplied message, the widget's actual class, and the expected class and calls **XtErrorMsg**. **XtCheckSubclass** should be used at the entry point of exported routines to ensure that the client has passed in a valid widget class for the exported operation.

XtCheckSubclass is only executed when the widget has been compiled with the compiler symbol DEBUG defined; otherwise, it is defined as the empty string and generates no code.

The **XtIsComposite** function is a convenience function that is equivalent to **XtIsSubclass** with **compositeWidgetClass** specified.

The **XtIsManaged** macro (for widget programmers) or function (for application programmers) returns **True** if the specified child widget is managed or **False** if it is not.

SEE ALSO

XtAppErrorMsg(3), XtDisplay(3)
X Toolkit Intrinsics - C Language Interface

Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtConfigureWidget, XtMoveWidget, XtResizeWidget - move and resize widgets

SYNTAX

void XtConfigureWidget(*w, x, y, width, height, border_width*)
Widget *w*;
Position *x*;
Position *y*;
Dimension *width*;
Dimension *height*;
Dimension *border_width*;

void XtMoveWidget(*w, x, y*)
Widget *w*;
Position *x*;
Position *y*;

void XtResizeWidget(*w, width, height, border_width*)
Widget *w*;
Dimension *width*;
Dimension *height*;
Dimension *border_width*;

void XtResizeWindow(*w*)
Widget *w*;

ARGUMENTS

width *height* *border_width*	Specify the new widget size.
w	Specifies the widget.
x *y*	Specify the new widget x and y coordinates.

DESCRIPTION

The **XtConfigureWidget** function returns immediately if the specified geometry fields are the same as the old values. Otherwise, **XtConfigureWidget** writes the new x, y, width, height, and border_width values into the widget and, if the widget is realized, makes an Xlib **XConfigureWindow** call on the widget's window.

If either the new width or height is different from its old value, **XtConfigureWidget** calls the widget's resize procedure to notify it of the size change; otherwise, it simply returns.

The **XtMoveWidget** function returns immediately if the specified geometry fields are the same as the old values. Otherwise, **XtMoveWidget** writes the new x and y values into the widget and, if the widget is realized, issues an Xlib **XMoveWindow** call on the widget's window.

The **XtResizeWidget** function returns immediately if the specified geometry fields are the same as the old values. Otherwise, **XtResizeWidget** writes the new width, height, and border_width values into the widget and, if the widget is realized, issues an **XConfigureWindow** call on the widget's window.

If the new width or height are different from the old values, **XtResizeWidget** calls the widget's resize procedure to notify it of the size change.

The **XtResizeWindow** function calls the **XConfigureWindow** Xlib function to make the window of the specified widget match its width, height, and border width. This request is done unconditionally because there is no way to tell if these values match the current values. Note that the widget's resize procedure is not called.

There are very few times to use **XtResizeWindow**; instead, you should use **XtResizeWidget**.

SEE ALSO

XtMakeGeometryRequest(3), XtQueryGeometry(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtConvert, XtDirectConvert – invoke resource converters

SYNTAX

void XtConvert(*w, from_type, from, to_type, to_return*)
Widget *w*;
String *from_type*;
XrmValuePtr *from*;
String *to_type*;
XrmValuePtr *to_return*;

void XtDirectConvert(*converter, args, num_args, from, to_return*)
XtConverter *converter*;
XrmValuePtr *args*;
Cardinal *num_args*;
XrmValuePtr *from*;
XrmValuePtr *to_return*;

ARGUMENTS

args	Specifies the argument list that contains the additional arguments needed to perform the conversion (often NULL).
converter	Specifies the conversion procedure that is to be called.
from	Specifies the value to be converted.
from_type	Specifies the source type.
num_args	Specifies the number of additional arguments (often zero).
to_type	Specifies the destination type.
to_return	Returns the converted value.
w	Specifies the widget to use for additional arguments (if any are needed).

DESCRIPTION

The **XtConvert** function looks up the type converter registered to convert from_type to to_type, computes any additional arguments needed, and then calls **XtDirectConvert**.

The **XtDirectConvert** function looks in the converter cache to see if this conversion procedure has been called with the specified arguments. If so, it returns a descriptor for information stored in the cache; otherwise, it calls the converter and enters the result in the cache.

Before calling the specified converter, **XtDirectConvert** sets the return value size to zero and the return value address to NULL. To determine if the conversion was successful, the client should check to_return.address for non-NULL.

SEE ALSO

XtAppAddConverter(3), XtStringConversionWarning(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtCreateApplicationContext, XtDestroyApplicationContext, XtWidgetToApplicationContext, XtToolkitInitialize - create, destroy, and obtain an application context

SYNTAX

XtAppContext XtCreateApplicationContext()

void XtDestroyApplicationContext(*app_context*)
XtAppContext *app_context*;

XtAppContext XtWidgetToApplicationContext(*w*)
Widget *w*;

void XtToolkitInitialize()

ARGUMENTS

app_context	Specifies the application context.
w	Specifies the widget to use for additional arguments (if any are needed).

DESCRIPTION

The **XtCreateApplicationContext** function returns an application context, which is an opaque type. Every application must have at least one application context.

The **XtDestroyApplicationContext** function destroys the specified application context as soon as it is safe to do so. If called from with an event dispatch (for example, a callback procedure), **XtDestroyApplicationContext** does not destroy the application context until the dispatch is complete.

The **XtWidgetToApplicationContext** function returns the application context for the specified widget.

The semantics of calling **XtToolkitInitialize** more than once are undefined.

SEE ALSO

XtDisplayInitialize(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtCreatePopupShell – attach a shell widget to the parent's pop-ups list directly

SYNTAX

Widget XtCreatePopupShell(*name, widget_class, parent, args, num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

ARGUMENTS

args	Specifies the argument list to override the resource defaults.
name	Specifies the text name for the created shell widget.
num_args	Specifies the number of arguments in the argument list.
parent	Specifies the parent widget.
widget_class	Specifies the widget class pointer for the created shell widget.

DESCRIPTION

The **XtCreatePopupShell** function ensures that the specified class is a subclass of **Shell** and, rather than using insert_child to attach the widget to the parent's children list, attaches the shell to the parent's pop-ups list directly.

A spring-loaded pop-up invoked from a translation table already must exist at the time that the translation is invoked, so the translation manager can find the shell by name. Pop-ups invoked in other ways can be created "on-the-fly" when the pop-up actually is needed. This delayed creation of the shell is particularly useful when you pop up an unspecified number of pop-ups. You can look to see if an appropriate unused shell (that is, not currently popped up) exists and create a new shell if needed.

SEE ALSO

XtCreateWidget(3), XtPopdown(3), XtPopup(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtCreateWidget, XtCreateManagedWidget, XtDestroyWidget – create and destroy widgets

SYNTAX

Widget XtCreateWidget(*name, widget_class, parent, args, num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

Widget XtCreateManagedWidget(*name, widget_class, parent, args, num_args*)
String *name*;
WidgetClass *widget_class*;
Widget *parent*;
ArgList *args*;
Cardinal *num_args*;

void XtDestroyWidget(*w*)
Widget *w*;

ARGUMENTS

args	Specifies the argument list to override the resource defaults.
name	Specifies the resource name for the created widget, which is used for retrieving resources and, for that reason, should not be the same as any other widget that is a child of same parent.
num_args	Specifies the number of arguments in the argument list.
parent	Specifies the parent widget.
w	Specifies the widget.
widget_class	Specifies the widget class pointer for the created widget.

DESCRIPTION

The **XtCreateWidget** function performs much of the boilerplate operations of widget creation:

- Checks to see if the class_initialize procedure has been called for this class and for all superclasses and, if not, calls those necessary in a superclass-to-subclass order.
- Allocates memory for the widget instance.
- If the parent is a subclass of **constraintWidgetClass**, it allocates memory for the parent's constraints and stores the address of this memory into the constraints field.
- Initializes the core nonresource data fields (for example, parent and visible).
- Initializes the resource fields (for example, background_pixel) by using the resource lists specified for this class and all superclasses.
- If the parent is a subclass of **constraintWidgetClass**, it initializes the resource fields of the constraints record by using the constraint resource list specified for the parent's class and all superclasses up to **constraintWidgetClass**.
- Calls the initialize procedures for the widget by starting at the **Core** initialize procedure on down to the widget's initialize procedure.
- If the parent is a subclass of **compositeWidgetClass**, it puts the widget into its parent's children list by calling its parent's insert_child procedure. For further information, see Section 3.5.
- If the parent is a subclass of **constraintWidgetClass**, it calls the constraint initialize procedures, starting at **constraintWidgetClass** on down to the parent's constraint initialize

procedure.

Note that you can determine the number of arguments in an argument list by using the **XtNumber** macro. For further information, see Section 11.1.

The **XtCreateManagedWidget** function is a convenience routine that calls **XtCreateWidget** and **XtManageChild**.

The **XtDestroyWidget** function provides the only method of destroying a widget, including widgets that need to destroy themselves. It can be called at any time, including from an application callback routine of the widget being destroyed. This requires a two-phase destroy process in order to avoid dangling references to destroyed widgets.

In phase one, **XtDestroyWidget** performs the following:

- If the being_destroyed field of the widget is **True**, it returns immediately.
- Recursively descends the widget tree and sets the being_destroyed field to **True** for the widget and all children.
- Adds the widget to a list of widgets (the destroy list) that should be destroyed when it is safe to do so.

Entries on the destroy list satisfy the invariant that if w2 occurs after w1 on the destroy list then w2 is not a descendent of w1. (A descendant refers to both normal and pop-up children.)

Phase two occurs when all procedures that should execute as a result of the current event have been called (including all procedures registered with the event and translation managers), that is, when the current invocation of **XtDispatchEvent** is about to return or immediately if not in **XtDispatchEvent**.

In phase two, **XtDestroyWidget** performs the following on each entry in the destroy list:

- Calls the destroy callback procedures registered on the widget (and all descendants) in post-order (it calls children callbacks before parent callbacks).
- If the widget's parent is a subclass of **compositeWidgetClass** and if the parent is not being destroyed, it calls **XtUnmanageChild** on the widget and then calls the widget's parent's delete_child procedure (see Section 3.4).
- If the widget's parent is a subclass of **constraintWidgetClass**, it calls the constraint destroy procedure for the parent, then the parent's superclass, until finally it calls the constraint destroy procedure for **constraintWidgetClass**.
- Calls the destroy methods for the widget (and all descendants) in post-order. For each such widget, it calls the destroy procedure declared in the widget class, then the destroy procedure declared in its superclass, until finally it calls the destroy procedure declared in the Core class record.
- Calls **XDestroyWindow** if the widget is realized (that is, has an X window). The server recursively destroys all descendant windows.
- Recursively descends the tree and deallocates all pop-up widgets, constraint records, callback lists and, if the widget is a subclass of **compositeWidgetClass**, children.

SEE ALSO

XtAppCreateShell(3), XtCreatePopupShell(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtCreateWindow – window creation convenience function

SYNTAX

void XtCreateWindow(*w, window_class, visual, value_mask, attributes*)
Widget *w*;
unsigned int *window_class*;
Visual **visual*;
XtValueMask *value_mask*;
XSetWindowAttributes **attributes*;

ARGUMENTS

attributes	Specifies the window attributes to use in the **XCreateWindow** call.
value_mask	Specifies which attribute fields to use.
visual	Specifies the visual type (usually **CopyFromParent**).
w	Specifies the widget that is used to set the x,y coordinates and so on.
window_class	Specifies the Xlib window class (for example, **InputOutput**, **InputOnly**, or **CopyFromParent**).

DESCRIPTION

The **XtCreateWindow** function calls the Xlib **XCreateWindow** function with values from the widget structure and the passed parameters. Then, it assigns the created window to the widget's window field.

XtCreateWindow evaluates the following fields of the **Core** widget structure:

- depth
- screen
- parent -> core.window
- x
- y
- width
- height
- border_width

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtDisplay, XtParent, XtScreen, XtWindow – obtain window information about a widget

SYNTAX

```
Display *XtDisplay(w)
    Widget w;

Widget XtParent(w)
    Widget w;

Screen *XtScreen(w)
    Widget w;

Window XtWindow(w)
    Widget w;
```

ARGUMENTS

w	Specifies the widget.

DESCRIPTION

XtDisplay returns the display pointer for the specified widget.

XtParent returns the parent widget for the specified widget.

XtScreen returns the screen pointer for the specified widget.

XtWindow returns the window of the specified widget.

SEE ALSO

XtClass(3)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtDisplayInitialize, XtOpenDisplay, XtDatabase, XtCloseDisplay – initialize, open, or close a display

SYNTAX

void XtToolkitInitialize()

void XtDisplayInitialize(*app_context, display, application_name, application_class, options, num_options, argc, argv*)
XtAppContext *app_context*;
Display **display*;
String *application_name*;
String *application_class*;
XrmOptionDescRec **options*;
Cardinal *num_options*;
Cardinal **argc*;
String **argv*;

Display *XtOpenDisplay(*app_context, display_string, application_name, application_class, options, num_options, argc, argv*)
XtAppContext *app_context*;
String *display_string*;
String *application_name*;
String *application_class*;
XrmOptionDescRec **options*;
Cardinal *num_options*;
Cardinal **argc*;
String **argv*;

void XtCloseDisplay(*display*)
Display **display*;

XrmDatabase XtDatabase(*display*)
Display **display*;

ARGUMENTS

argc Specifies a pointer to the number of command line parameters.

argv Specifies the command line parameters.

app_context Specifies the application context.

application_class Specifies the class name of this application, which usually is the generic name for all instances of this application.

application_name Specifies the name of the application instance.

display Specifies the display from which to get the resources. Note that a display can be in at most one application context.

num_options Specifies the number of entries in the options list.

options Specifies how to parse the command line for any application-specific resources. The options argument is passed as a parameter to **XrmParseCommand**. For further information, see *Xlib - C Language X Interface.*

DESCRIPTION

The **XtDisplayInitialize** function builds the resource database, calls the Xlib **XrmParseCommand** function to parse the command line, and performs other per display initialization. After **XrmParseCommand** has been called, argc and argv contain only those parameters that were

not in the standard option table or in the table specified by the options argument. If the modified argc is not zero, most applications simply print out the modified argv along with a message listing the allowable options. On UNIX-based systems, the application name is usually the final component of argv[0]. If the synchronize resource is **True** for the specified application, **XtDisplayInitialize** calls the Xlib **XSynchronize** function to put Xlib into synchronous mode for this display connection. If the reverseVideo resource is **True**, the Intrinsics exchange **XtDefaultForeground** and **XtDefaultBackground** for widgets created on this display. (See Section 9.6.1).

The **XtOpenDisplay** function calls **XOpenDisplay** the specified display name. If display_string is NULL, **XtOpenDisplay** uses the current value of the -display option specified in argv and if no display is specified in argv, uses the user's default display (on UNIX-based systems, this is the value of the DISPLAY environment variable).

If this succeeds, it then calls **XtDisplayInitialize** and pass it the opened display and the value of the -name option specified in argv as the application name. If no name option is specified, it uses the application name passed to **XtOpenDisplay**. If the application name is NULL, it uses the last component of argv[0]. **XtOpenDisplay** returns the newly opened display or NULL if it failed.

XtOpenDisplay is provided as a convenience to the application programmer.

The **XtCloseDisplay** function closes the specified display as soon as it is safe to do so. If called from within an event dispatch (for example, a callback procedure), **XtCloseDisplay** does not close the display until the dispatch is complete. Note that applications need only call **XtCloseDisplay** if they are to continue executing after closing the display; otherwise, they should call **XtDestroyApplicationContext** or just exit.

The **XtDatabase** function returns the fully merged resource database that was built by **XtDisplayInitialize** associated with the display that was passed in. If this display has not been initialized by **XtDisplayInitialize**, the results are not defined.

SEE ALSO

XtAppCreateShell(3), XtCreateApplicationContext(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtGetGC, XtReleaseGC – obtain and destroy a sharable GC

SYNTAX

GC XtGetGC(*w*, *value_mask*, *values*)
Widget *w*;
XtGCMask *value_mask*;
XGCValues **values*;

void XtReleaseGC(*w*, *gc*)
Widget *w*;
GC *gc*;

ARGUMENTS

gc	Specifies the GC to be deallocated.
values	Specifies the actual values for this GC.
value_mask	Specifies which fields of the values are specified.
w	Specifies the widget.

DESCRIPTION

The **XtGetGC** function returns a sharable, read-only GC. The parameters to this function are the same as those for **XCreateGC** except that a widget is passed instead of a display. **XtGetGC** shares only GCs in which all values in the GC returned by **XCreateGC** are the same. In particular, it does not use the value_mask provided to determine which fields of the GC a widget considers relevant. The value_mask is used only to tell the server which fields should be filled in with widget data and which it should fill in with default values. For further information about value_mask and values, see **XCreateGC** in the *Xlib - C Language X Interface*.

The **XtReleaseGC** function deallocate the specified shared GC.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtGetResourceList – obtain resource list

SYNTAX

void XtGetResourceList(*class, resources_return, num_resources_return*);
WidgetClass *class*;
XtResourceList **resources_return*;
Cardinal **num_resources_return*;

ARGUMENTS

num_resources_return
Specifies a pointer to where to store the number of entries in the resource list.

resources_return
Specifies a pointer to where to store the returned resource list. The caller must free this storage using **XtFree** when done with it.

widget_class Specifies the widget class pointer for the created widget.

DESCRIPTION

If it is called before the widget class is initialized (that is, before the first widget of that class has been created), **XtGetResourceList** returns the resource list as specified in the widget class record. If it is called after the widget class has been initialized, **XtGetResourceList** returns a merged resource list that contains the resources for all superclasses.

SEE ALSO

XtGetSubresources(3), XtOffset(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtGetSelectionValue, XtGetSelectionValues – obtain selection values

SYNTAX

void XtGetSelectionValue(*w, selection, target, callback, client_data, time*)
Widget *w*;
Atom *selection*;
Atom *target*;
XtSelectionCallbackProc *callback*;
XtPointer *client_data*;
Time *time*;

void XtGetSelectionValues(*w, selection, targets, count, callback, client_data, time*)
Widget *w*;
Atom *selection*;
Atom **targets*;
int *count*;
XtSelectionCallbackProc *callback*;
XtPointer *client_data*;
Time *time*;

ARGUMENTS

callback	Specifies the callback procedure that is to be called when the selection value has been obtained.
client_data	Specifies the argument that is to be passed to the specified procedure when it is called.
client_data	Specifies the client data (one for each target type) that is passed to the callback procedure when it is called for that target.
count	Specifies the length of the targets and client_data lists.
selection	Specifies the particular selection desired (that is, primary or secondary).
target	Specifies the type of the information that is needed about the selection.
targets	Specifies the types of information that is needed about the selection.
time	Specifies the timestamp that indicates when the selection value is desired.
w	Specifies the widget that is making the request.

DESCRIPTION

The **XtGetSelectionValue** function requests the value of the selection that has been converted to the target type. The specified callback will be called some time after **XtGetSelectionValue** is called; in fact, it may be called before or after **XtGetSelectionValue** returns.

The **XtGetSelectionValues** function is similar to **XtGetSelectionValue** except that it takes a list of target types and a list of client data and obtains the current value of the selection converted to each of the targets. The effect is as if each target were specified in a separate call to **XtGetSelectionValue**. The callback is called once with the corresponding client data for each target. **XtGetSelectionValues** does guarantee that all the conversions will use the same selection value becaues the ownership of the selection cannot change in the middle of the list, as would be when calling **XtGetSelectionValue** repeatedly.

SEE ALSO

XtAppGetSelectionTimeout(3), XtOwnSelection(3)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtGetSubresources, XtGetApplicationResources - obtain subresources or application resources

SYNTAX

void XtGetSubresources(*w, base, name, class, resources, num_resources, args, num_args*)
Widget *w*;
XtPointer *base*;
String *name*;
String *class*;
XtResourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

void XtGetApplicationResources(*w, base, resources, num_resources, args, num_args*)
Widget *w*;
XtPointer *base*;
XtResourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

ARGUMENTS

args	Specifies the argument list to override resources obtained from the resource database.
base	Specifies the base address of the subpart data structure where the resources should be written.
class	Specifies the class of the subpart.
name	Specifies the name of the subpart.
num_args	Specifies the number of arguments in the argument list.
num_resources	Specifies the number of resources in the resource list.
resources	Specifies the resource list for the subpart.
w	Specifies the widget that wants resources for a subpart or that identifies the resource database to search.

DESCRIPTION

The **XtGetSubresources** function constructs a name/class list from the application name/class, the name/classes of all its ancestors, and the widget itself. Then, it appends to this list the name/class pair passed in. The resources are fetched from the argument list, the resource database, or the default values in the resource list. Then, they are copied into the subpart record. If args is NULL, num_args must be zero. However, if num_args is zero, the argument list is not referenced.

The **XtGetApplicationResources** function first uses the passed widget, which is usually an application shell, to construct a resource name and class list, Then, it retrieves the resources from the argument list, the resource database, or the resource list default values. After adding base to each address, **XtGetApplicationResources** copies the resources into the address given in the resource list. If args is NULL, num_args must be zero. However, if num_args is zero, the argument list is not referenced. The portable way to specify application resources is to declare them as members of a structure and pass the address of the structure as the base argument.

SEE ALSO

XtGetResourceList(3)
X Toolkit Intrinsics - C Language Interface

Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtMakeGeometryRequest, XtMakeResizeRequest – make geometry manager request

SYNTAX

XtGeometryResult XtMakeGeometryRequest(*w, request, reply_return*)
Widget *w*;
XtWidgetGeometry **request*;
XtWidgetGeometry **reply_return*;

XtGeometryResult XtMakeResizeRequest(*w, width, height, width_return, height_return*)
Widget *w*;
Dimension *width, height*;
Dimension **width_return*, **height_return*

ARGUMENTS

reply_return	Returns the allowed widget size or may be NULL if the requesting widget is not interested in handling **XtGeometryAlmost**.
request	Specifies the desired widget geometry (size, position, border width, and stacking order).
w	Specifies the widget that is making the request.
width_return *height_return*	Return the allowed widget width and height.

DESCRIPTION

Depending on the condition, **XtMakeGeometryRequest** performs the following:

- If the widget is unmanaged or the widget's parent is not realized, it makes the changes and returns **XtGeometryYes**.
- If the parent is not a subclass of **compositeWidgetClass** or the parent's geometry_manager is NULL, it issues an error.
- If the widget's being_destroyed field is **True**, it returns **XtGeometryNo**.
- If the widget x, y, width, height and border_width fields are all equal to the requested values, it returns **XtGeometryYes**; otherwise, it calls the parent's geometry_manager procedure with the given parameters.
- If the parent's geometry manager returns **XtGeometryYes** and if **XtCWQueryOnly** is not set in the request_mode and if the widget is realized, **XtMakeGeometryRequest** calls the **XConfigureWindow** Xlib function to reconfigure the widget's window (set its size, location, and stacking order as appropriate).
- If the geometry manager returns **XtGeometryDone**, the change has been approved and actually has been done. In this case, **XtMakeGeometryRequest** does no configuring and returns **XtGeometryYes**. **XtMakeGeometryRequest** never returns **XtGeometryDone**.

Otherwise, **XtMakeGeometryRequest** returns the resulting value from the parent's geometry manager.

Children of primitive widgets are always unmanaged; thus, **XtMakeGeometryRequest** always returns **XtGeometryYes** when called by a child of a primitive widget.

The **XtMakeResizeRequest** function, a simple interface to **XtMakeGeometryRequest**, creates a **XtWidgetGeometry** structure and specifies that width and height should change. The geometry manager is free to modify any of the other window attributes (position or stacking order) to satisfy the resize request. If the return value is **XtGeometryAlmost**, width_return and height_return contain a compromise width and height. If these are acceptable, the widget should immediately make an **XtMakeResizeRequest** and request that the compromise width

and height be applied. If the widget is not interested in **XtGeometryAlmost** replies, it can pass NULL for width_return and height_return.

SEE ALSO

XtConfigureWidget(3), XtQueryGeometery(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtMalloc, XtCalloc, XtRealloc, XtFree, XtNew, XtNewString – memory management functions

SYNTAX

char *XtMalloc(*size*);
Cardinal *size*;

char *XtCalloc(*num*, *size*);
Cardinal *num*;
Cardinal *size*;

char *XtRealloc(*ptr*, *num*);
char **ptr*;
Cardinal *num*;

void XtFree(*ptr*);
char **ptr*;

type *XtNew(*type*);
type;

String XtNewString(*string*);
String *string*;

ARGUMENTS

num	Specifies the number of bytes or array elements.
ptr	Specifies a pointer to the old storage or to the block of storage that is to be freed.
size	Specifies the size of an array element (in bytes) or the number of bytes desired.
string	Specifies a previously declared string.
type	Specifies a previously declared data type.

DESCRIPTION

The **XtMalloc** functions returns a pointer to a block of storage of at least the specified size bytes. If there is insufficient memory to allocate the new block, **XtMalloc** calls **XtErrorMsg**.

The **XtCalloc** function allocates space for the specified number of array elements of the specified size and initializes the space to zero. If there is insufficient memory to allocate the new block, **XtCalloc** calls **XtErrorMsg**.

The **XtRealloc** function changes the size of a block of storage (possibly moving it). Then, it copies the old contents (or as much as will fit) into the new block and frees the old block. If there is insufficient memory to allocate the new block, **XtRealloc** calls **XtErrorMsg**. If ptr is NULL, **XtRealloc** allocates the new storage without copying the old contents; that is, it simply calls **XtMalloc**.

The **XtFree** function returns storage and allows it to be reused. If ptr is NULL, **XtFree** returns immediately.

XtNew returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, **XtNew** calls **XtErrorMsg**. **XtNew** is a convenience macro that calls **XtMalloc** with the following arguments specified:

((type *) XtMalloc((unsigned) sizeof(type))

XtNewString returns a pointer to the allocated storage. If there is insufficient memory to allocate the new block, **XtNewString** calls **XtErrorMsg**. **XtNewString** is a convenience macro that calls **XtMalloc** with the following arguments specified:

(strcpy(XtMalloc((unsigned) strlen(str) + 1), str))

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtManageChildren, XtManageChild, XtUnmanageChildren, XtUnmanageChild – manage and unmanage children

SYNTAX

typedef Widget *WidgetList;

void XtManageChildren(*children, num_children*)
WidgetList *children*;
Cardinal *num_children*;

void XtManageChild(*child*)
Widget *child*;

void XtUnmanageChildren(*children, num_children*)
WidgetList *children*;
Cardinal *num_children*;

void XtUnmanageChild(*child*)
Widget *child*;

ARGUMENTS

child	Specifies the child.
children	Specifies a list of child widgets.
num_childrcn	Specifies the number of children.

DESCRIPTION

The **XtManageChildren** function performs the following:

- Issues an error if the children do not all have the same parent or if the parent is not a subclass of **compositeWidgetClass**.
- Returns immediately if the common parent is being destroyed; otherwise, for each unique child on the list, **XtManageChildren** ignores the child if it already is managed or is being destroyed and marks it if not.
- If the parent is realized and after all children have been marked, it makes some of the newly managed children viewable:
  - Calls the change_managed routine of the widgets' parent.
  - Calls **XtRealizeWidget** on each previously unmanaged child that is unrealized.
  - Maps each previously unmanaged child that has map_when_managed **True**.

Managing children is independent of the ordering of children and independent of creating and deleting children. The layout routine of the parent should consider children whose managed field is **True** and should ignore all other children. Note that some composite widgets, especially fixed boxes, call **XtManageChild** from their insert_child procedure.

If the parent widget is realized, its change_managed procedure is called to notify it that its set of managed children has changed. The parent can reposition and resize any of its children. It moves each child as needed by calling **XtMoveWidget**, which first updates the x and y fields and then calls **XMoveWindow** if the widget is realized.

The **XtManageChild** function constructs a **WidgetList** of length one and calls **XtManageChildren**.

The **XtUnmanageChildren** function performs the following:

- Issues an error if the children do not all have the same parent or if the parent is not a subclass of **compositeWidgetClass**.

- Returns immediately if the common parent is being destroyed; otherwise, for each unique child on the list, **XtUnmanageChildren** performs the following:
  - Ignores the child if it already is unmanaged or is being destroyed and marks it if not.
  - If the child is realized, it makes it nonvisible by unmapping it.
- Calls the change_managed routine of the widgets' parent after all children have been marked if the parent is realized.

XtUnmanageChildren does not destroy the children widgets. Removing widgets from a parent's managed set is often a temporary banishment, and, some time later, you may manage the children again.

The **XtUnmanageChild** function constructs a widget list of length one and calls **XtUnmanageChildren**.

SEE ALSO

XtMapWidget(3), XtRealizeWidget(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtMapWidget, XtSetMappedWhenManaged, XtUnmapWidget – map and unmap widgets

SYNTAX

XtMapWidget(*w*)
Widget *w*;

void XtSetMappedWhenManaged(*w*, *map_when_managed*)
Widget *w*;
Boolean *map_when_managed*;

XtUnmapWidget(*w*)
Widget *w*;

ARGUMENTS

map_when_managed
Specifies a Boolean value that indicates the new value of the map_when_managed field.

w Specifies the widget.

DESCRIPTION

If the widget is realized and managed and if the new value of map_when_managed is **True**, **XtSetMappedWhenManaged** maps the window. If the widget is realized and managed and if the new value of map_when_managed is **False**, it unmaps the window. **XtSetMappedWhenManaged** is a convenience function that is equivalent to (but slightly faster than) calling **XtSetValues** and setting the new value for the mappedWhenManaged resource. As an alternative to using **XtSetMappedWhenManaged** to control mapping, a client may set mapped_when_managed to **False** and use **XtMapWidget** and **XtUnmapWidget** explicitly.

SEE ALSO

XtManageChildren(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtNameToWidget, XtWidgetToWindow – translating strings to widgets or widgets to windows

SYNTAX

Widget XtNameToWidget(*reference, names*);
Widget *reference*;
String *names*;

Widget XtWindowToWidget(*display, window*)
Display **display*;
Window *window*;

ARGUMENTS

display	Specifies the display on which the window is defined.
names	Specifies the fully qualified name of the desired widget.
reference	Specifies the widget from which the search is to start.
window	Specify the window for which you want the widget.

DESCRIPTION

The **XtNameToWidget** function looks for a widget whose name is the first component in the specified names and that is a pop-up child of reference (or a normal child if reference is a subclass of **compositeWidgetClass**). It then uses that widget as the new reference and repeats the search after deleting the first component from the specified names. If it cannot find the specified widget, **XtNameToWidget** returns NULL.

Note that the names argument contains the name of a widget with respect to the specified reference widget and can contain more than one widget name (separated by periods) for widgets that are not direct children of the specified reference widget.

If more than one child of the reference widget matches the name, **XtNameToWidget** can return any of the children. The Intrinsics do not require that all children of a widget have unique names. If the specified names contain more than one component and if more than one child matches the first component, **XtNameToWidget** can return NULL if the single branch that it follows does not contain the named widget. That is, **XtNameToWidget** does not back up and follow other matching branches of the widget tree.

The **XtWindowToWidget** function translates the specified window and display pointer into the appropriate widget instance.

SEE ALSO

X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtOffset, XtNumber – determine the byte offset or number of array elements

SYNTAX

Cardinal XtOffset(*pointer_type*, *field_name*)
    Type *pointer_type*;
    Field *field_name*;

Cardinal XtNumber(*array*)
    ArrayVariable *array*;

ARGUMENTS

array	Specifies a fixed-size array.
field_name	Specifies the name of the field for which to calculate the byte offset.
pointer_type	Specifies a type that is declared as a pointer to the structure.

DESCRIPTION

The **XtOffset** macro is usually used to determine the offset of various resource fields from the beginning of a widget and can be used at compile time in static initializations.

The **XtNumber** macro returns the number of elements in the specified argument lists, resources lists, and other counted arrays.

SEE ALSO

XtGetResourceList(3), XtSetArg(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtOwnSelection, XtDisownSelection – set selection owner

SYNTAX

Boolean XtOwnSelection(*w, selection, time, convert_proc, lose_selection, done_proc*)
Widget *w*;
Atom *selection*;
Time *time*;
XtConvertSelectionProc *convert_proc*;
XtLoseSelectionProc *lose_selection*;
XtSelectionDoneProc *done_proc*;

void XtDisownSelection(*w, selection, time*)
Widget *w*;
Atom *selection*;
Time *time*;

ARGUMENTS

convert_proc	Specifies the procedure that is to be called whenever someone requests the current value of the selection.
done_proc	Specifies the procedure that is called after the requestor has received the selection or NULL if the owner is not interested in being called back.
lose_selection	Specifies the procedure that is to be called whenever the widget has lost selection ownership or NULL if the owner is not interested in being called back.
selection	Specifies an atom that describes the type of the selection (for example, **XA_PRIMARY**, **XA_SECONDARY**, or **XA_CLIPBOARD**).
time	Specifies the timestamp that indicates when the selection ownership should commence or is to be relinquished.
w	Specifies the widget that wishes to become the owner or to relinquish ownership.

DESCRIPTION

The **XtOwnSelection** function informs the Intrinsics selection mechanism that a widget believes it owns a selection. It returns **True** if the widget has successfully become the owner and **False** otherwise. The widget may fail to become the owner if some other widget has asserted ownership at a time later than this widget. Note that widgets can lose selection ownership either because someone else asserted later ownership of the selection or because the widget voluntarily gave up ownership of the selection. Also note that the lose_selection procedure is not called if the widget fails to obtain selection ownership in the first place.

The **XtDisownSelection** function informs the Intrinsics selection mechanism that the specified widget is to lose ownership of the selection. If the widget does not currently own the selection either because it lost the selection or because it never had the selection to begin with, **XtDisownSelection** does nothing.

After a widget has called **XtDisownSelection**, its convert procedure is not called even if a request arrives later with a timestamp during the period that this widget owned the selection. However, its done procedure will be called if a conversion that started before the call to **XtDisownSelection** finishes after the call to **XtDisownSelection**.

SEE ALSO

XtAppGetSelectionTimeout(3), XtGetSelectionValue(3)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtParseAcceleratorTable, XtInstallAccelerators, XtInstallAllAccelerators - managing accelerator tables

SYNTAX

XtAccelerators XtParseAcceleratorTable(*source*)
String *source*;

void XtInstallAccelerators(*destination, source*)
Widget *destination*;
Widget *source*;

void XtInstallAllAccelerators(*destination, source*)
Widget *destination*;
Widget *source*;

ARGUMENTS

source	Specifies the accelerator table to compile.
destination	Specifies the widget on which the accelerators are to be installed.
source	Specifies the widget or the root widget of the widget tree from which the accelerators are to come.

DESCRIPTION

The **XtParseAcceleratorTable** function compiles the accelerator table into the opaque internal representation.

The **XtInstallAccelerators** function installs the accelerators from source onto destination by augmenting the destination translations with the source accelerators. If the source display_accelerator method is non-NULL, **XtInstallAccelerators** calls it with the source widget and a string representation of the accelerator table, which indicates that its accelerators have been installed and that it should display them appropriately. The string representation of the accelerator table is its canonical translation table representation.

The **XtInstallAllAccelerators** function recursively descends the widget tree rooted at source and installs the accelerators of each widget encountered onto destination. A common use os to call **XtInstallAllAccelerators** and pass the application main window as the source.

SEE ALSO

XtParseTranslationTable(1)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtParseTranslationTable, XtAugmentTranslations, XtOverrideTranslations, XtUninstallTranslations – manage translation tables

SYNTAX

XtTranslations XtParseTranslationTable(*table*)
String *table*;

void XtAugmentTranslations(*w*, *translations*)
Widget *w*;
XtTranslations *translations*;

void XtOverrideTranslations(*w*, *translations*)
Widget *w*;
XtTranslations *translations*;

void XtUninstallTranslations(*w*)
Widget *w*;

ARGUMENTS

table	Specifies the translation table to compile.
translations	Specifies the compiled translation table to merge in (must not be NULL).
w	Specifies the widget into which the new translations are to be merged or removed.

DESCRIPTION

The **XtParseTranslationTable** function compiles the translation table into the opaque internal representation of type **XtTranslations**. Note that if an empty translation table is required for any purpose, one can be obtained by calling **XtParseTranslationTable** and passing an empty string.

The **XtAugmentTranslations** function nondestructively merges the new translations into the existing widget translations. If the new translations contain an event or event sequence that already exists in the widget's translations, the new translation is ignored.

The **XtOverrideTranslations** function destructively merges the new translations into the existing widget translations. If the new translations contain an event or event sequence that already exists in the widget's translations, the new translation is merged in and override the widget's translation.

To replace a widget's translations completely, use **XtSetValues** on the XtNtranslations resource and specifiy a compiled translation table as the value.

The **XtUninstallTranslations** function causes the entire translation table for widget to be removed.

SEE ALSO

XtAppAddActions(3), XtCreatePopupShell(3), XtParseAcceleratorTable(3), XtPopup(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtPopdown, XtCallbackPopdown, MenuPopdown – unmap a pop-up

SYNTAX

void XtPopdown(*popup_shell*)
Widget *popup_shell*;

void XtCallbackPopdown(*w*, *client_data*, *call_data*)
Widget *w*;
XtPointer *client_data*;
XtPointer *call_data*;

void MenuPopdown(*shell_name*)
String *shell_name*;

ARGUMENTS

call_data	Specifies the callback data, which is not used by this procedure.
client_data	Specifies a pointer to the **XtPopdownID** structure.
popup_shell	Specifies the widget shell to pop down.
shell_name	Specifies the name of the widget shell to pop down.
w	Specifies the widget.

DESCRIPTION

The **XtPopdown** function performs the following:

- Calls **XtCheckSubclass** to ensure popup_shell is a subclass of **Shell**.
- Checks that popup_shell is currently popped_up; otherwise, it generates an error.
- Unmaps popup_shell's window.
- If popup_shell's grab_kind is either **XtGrabNonexclusive** or **XtGrabExclusive**, it calls **XtRemoveGrab**.
- Sets pop-up shell's popped_up field to **False**.
- Calls the callback procedures on the shell's popdown_callback list.

The **XtCallbackPopdown** function casts the client data parameter to an **XtPopdownID** pointer:

```
typedef struct {
        Widget shell_widget;
        Widget enable_widget;
} XtPopdownIDRec, *XtPopdownID;
```

The shell_widget is the pop-up shell to pop down, and the enable_widget is the widget that was used to pop it up.

XtCallbackPopdown calls **XtPopdown** with the specified shell_widget and then calls **XtSetSensitive** to resensitize the enable_widget.

If a shell name is not given, **MenuPopdown** calls **XtPopdown** with the widget for which the translation is specified. If a shell_name is specified in the translation table, **MenuPopdown** tries to find the shell by looking up the widget tree starting at the parent of the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops down the shell; otherwise, it moves up the parent chain as needed. If **MenuPopdown** gets to the application top-level shell widget and cannot find a matching shell, it generates an error.

SEE ALSO

XtCreatePopupShell(3), XtPopup(3)
X Toolkit Intrinsics – C Language Interface

Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtPopup, XtCallbackNone, XtCallbackNonexclusive, XtCallbackExclusive, MenuPopup – map a pop-up

SYNTAX

void XtPopup(*popup_shell, grab_kind*)
Widget *popup_shell*;
XtGrabKind *grab_kind*;

void XtCallbackNone(*w, client_data, call_data*)
Widget *w*;
XtPointer *client_data*;
XtPointer *call_data*;

void XtCallbackNonexclusive(*w, client_data, call_data*)
Widget *w*;
XtPointer *client_data*;
XtPointer *call_data*;

void XtCallbackExclusive(*w, client_data, call_data*)
Widget *w*;
XtPointer *client_data*;
XtPointer *call_data*;

void MenuPopup(*shell_name*)
String *shell_name*;

ARGUMENTS

call_data	Specifies the callback data, which is not used by this procedure.
client_data	Specifies the pop-up shell.
grab_kind	Specifies the way in which user events should be constrained.
popup_shell	Specifies the widget shell to pop down.
w	Specifies the widget.

DESCRIPTION

The **XtPopup** function performs the following:

- Calls **XtCheckSubclass** to ensure popup_shell is a subclass of **Shell**.
- Generates an error if the shell's popped_up field is already **True**.
- Calls the callback procedures on the shell's popup_callback list.
- Sets the shell popped_up field to **True**, the shell spring_loaded field to **False**, and the shell grab_kind field from grab_kind.
- If the shell's create_popup_child field is non-NULL, **XtPopup** calls it with popup_shell as the parameter.
- If grab_kind is either **XtGrabNonexclusive** or **XtGrabExclusive**, it calls:

XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), False)

- Calls **XtRealizeWidget** with popup_shell specified.
- Calls **XMapWindow** with popup_shell specified.

The **XtCallbackNone**, **XtCallbackNonexclusive**, and **XtCallbackExclusive** functions call **XtPopup** with the shell specified by the client data argument and grab_kind set as the name specifies. **XtCallbackNone**, **XtCallbackNonexclusive**, and **XtCallbackExclusive** specify **XtGrabNone**, **XtGrabNonexclusive**, and **XtGrabExclusive**, respectively. Each function then sets the widget that executed the callback list to be insensitive by using **XtSetSensitive**.

Using these functions in callbacks is not required. In particular, an application must provide customized code for callbacks that create pop-up shells dynamically or that must do more than desensitizing the button.

MenuPopup is known to the translation manager, which must perform special actions for spring-loaded pop-ups. Calls to **MenuPopup** in a translation specification are mapped into calls to a nonexported action procedure, and the translation manager fills in parameters based on the event specified on the left-hand side of a translation.

If **MenuPopup** is invoked on **ButtonPress** (possibly with modifiers), the translation manager pops up the shell with grab_kind set to **XtGrabExclusive** and spring_loaded set to **True**. If **MenuPopup** is invoked on **EnterWindow** (possibly with modifiers), the translation manager pops up the shell with grab_kind set to **XtGrabNonexclusive** and spring_loaded set to **False**. Otherwise, the translation manager generates an error. When the widget is popped up, the following actions occur:

- Calls **XtCheckSubclass** to ensure popup_shell is a subclass of **Shell**.
- Generates an error if the shell's popped_up field is already **True**.
- Calls the callback procedures on the shell's popup_callback list.
- Sets the shell popped_up field to **True** and the shell grab_kind and spring_loaded fields appropriately.
- If the shell's create_popup_child field is non-NULL, it is called with popup_shell as the parameter.
- Calls:

XtAddGrab(popup_shell, (grab_kind == XtGrabExclusive), spring_loaded)

- Calls **XtRealizeWidget** with popup_shell specified.
- Calls **XMapWindow** with popup_shell specified.

(Note that these actions are the same as those for **XtPopup**.) **MenuPopup** tries to find the shell by searching the widget tree starting at the parent of the widget in which it is invoked. If it finds a shell with the specified name in the pop-up children of that parent, it pops up the shell with the appropriate parameters. Otherwise, it moves up the parent chain as needed. If **MenuPopup** gets to the application widget and cannot find a matching shell, it generates an error.

SEE ALSO

XtCreatePopupShell(3), XtPopdown(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtQueryGeometry – query the preferred geometry of a child widget

SYNTAX

XtGeometryResult XtQueryGeometry(*w, intended, preferred_return*)
Widget *w*;
XtWidgetGeometry **intended*, **preferred_return*;

ARGUMENTS

intended Specifies any changes the parent plans to make to the child's geometry or NULL.

preferred_return
Returns the child widget's preferred geometry.

w Specifies the widget.

DESCRIPTION

To discover a child's preferred geometry, the child's parent sets any changes that it intends to make to the child's geometry in the corresponding fields of the intended structure, sets the corresponding bits in intended.request_mode, and calls **XtQueryGeometry**.

XtQueryGeometry clears all bits in the preferred_return->request_mode and checks the query_geometry field of the specified widget's class record. If query_geometry is not NULL, **XtQueryGeometry** calls the query_geometry procedure and passes as arguments the specified widget, intended, and preferred_return structures. If the intended argument is NULL, **XtQueryGeometry** replaces it with a pointer to an **XtWidgetGeometry** structure with request_mode=0 before calling query_geometry.

SEE ALSO

XtConfigureWidget(3), XtMakeGeometryRequest(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtRealizeWidget, XtIsRealized, XtUnrealizeWidget - realize and unrealize widgets

SYNTAX

void XtRealizeWidget(*w*)
Widget *w*;

Boolean XtIsRealized(*w*)
Widget *w*;

void XtUnrealizeWidget(*w*)
Widget *w*;

ARGUMENTS

w Specifies the widget.

DESCRIPTION

If the widget is already realized, **XtRealizeWidget** simply returns. Otherwise, it performs the following:

- Binds all action names in the widget's translation table to procedures (see Section 10.1.2).
- Makes a post-order traversal of the widget tree rooted at the specified widget and calls the change_managed procedure of each composite widget that has one or more managed children.
- Constructs an **XSetWindowAttributes** structure filled in with information derived from the **Core** widget fields and calls the realize procedure for the widget, which adds any widget-specific attributes and creates the X window.
- If the widget is not a subclass of **compositeWidgetClass**, **XtRealizeWidget** returns; otherwise, it continues and performs the following:
  - Descends recursively to each of the widget's managed children and calls the realize procedures. Primitive widgets that instantiate children are responsible for realizing those children themselves.
  - Maps all of the managed children windows that have mapped_when_managed **True**. (If a widget is managed but mapped_when_managed is **False**, the widget is allocated visual space but is not displayed. Some people seem to like this to indicate certain states.)

If the widget is a top-level shell widget (that is, it has no parent), and mapped_when_managed is **True**, **XtRealizeWidget** maps the widget window.

The **XtIsRealized** function returns **True** if the widget has been realized, that is, if the widget has a nonzero X window ID.

Some widget procedures (for example, set_values) might wish to operate differently after the widget has been realized.

The **XtUnrealizeWidget** function destroys the windows of an existing widget and all of its children (recursively down the widget tree). To recreate the windows at a later time, call **XtRealizeWidget** again. If the widget was managed, it will be unmanaged automatically before its window is freed.

SEE ALSO

XtManageChildren(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtSetArg, XtMergeArgLists – set and merge ArgLists

SYNTAX

XtSetArg(*arg, name, value*)
Arg *arg*;
String *name*;
XtArgVal *value*;

ArgList XtMergeArgLists(*args1, num_args1, args2, num_args2*)
ArgList *args1*;
Cardinal *num_args1*;
ArgList *args2*;
Cardinal *num_args2*;

ARGUMENTS

arg	Specifies the name-value pair to set.
args1	Specifies the first **ArgList**.
args2	Specifies the second **ArgList**.
num_args1	Specifies the number of arguments in the first argument list.
num_args2	Specifies the number of arguments in the second argument list.
name	Specifies the name of the resource.
value	Specifies the value of the resource if it will fit in an **XtArgVal** or the address.

DESCRIPTION

The **XtSetArg** function is usually used in a highly stylized manner to minimize the probability of making a mistake; for example:

```
Arg args[20];
int n;

n = 0;
XtSetArg(args[n], XtNheight, 100);          n++;
XtSetArg(args[n], XtNwidth, 200);           n++;
XtSetValues(widget, args, n);
```

Alternatively, an application can statically declare the argument list and use **XtNumber**:

```
static Args args[] = {
        {XtNheight, (XtArgVal) 100},
        {XtNwidth, (XtArgVal) 200},
};
XtSetValues(Widget, args, XtNumber(args));
```

Note that you should not use auto-increment or auto-decrement within the first argument to **XtSetArg**. **XtSetArg** can be implemented as a macro that dereferences the first argument twice.

The **XtMergeArgLists** function allocates enough storage to hold the combined **ArgList** structures and copies them into it. Note that it does not check for duplicate entries. When it is no longer needed, free the returned storage by using **XtFree**.

SEE ALSO

XtOffset(3)
X Toolkit Intrinsics – C Language Interface
Xlib – C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtSetKeyTranslator, XtTranslateKeycode, XtRegisterCaseConverter, XtConvertCase – convert KeySym to KeyCodes

SYNTAX

void XtSetKeyTranslator(*display, proc*)
Display **display*;
XtKeyProc *proc*;

void XtTranslateKeycode(*display, keycode, modifiers, modifiers_return, keysym_return*)
Display **display*;
KeyCode *keycode*;
Modifiers *modifiers*;
Modifiers **modifiers_return*;
KeySym **keysym_return*;

void XtRegisterCaseConverter(*display, proc, start, stop*)
Display **display*;
XtCaseProc *proc*;
KeySym *start*;
KeySym *stop*;

void XtConvertCase(*display, keysym, lower_return, upper_return*)
Display **display*;
KeySym *keysym*;
KeySym **lower_return*;
KeySym **upper_return*;

ARGUMENTS

display	Specifies the display.
keycode	Specifies the KeyCode to translate.
keysym	Specifies the KeySym to convert.
keysym_return	Returns the resulting KeySym.
lower_return	Returns the lowercase equivalent of the KeySym.
upper_return	Returns the uppercase equivalent of the KeySym.
modifiers	Specifies the modifiers to the KeyCode.
modifiers_return	Returns a mask that indicates the modifiers actually used to generate the KeySym.
proc	Specifies the procedure that is to perform key translations or conversions.
start	Specifies the first KeySym for which this converter is valid.
stop	Specifies the last KeySym for which this converter is valid.

DESCRIPTION

The **XtSetKeyTranslator** function sets the specified procedure as the current key translator. The default translator is **XtTranslateKey**, an **XtKeyProc** that uses Shift and Lock modifiers with the interpretations defined by the core protocol. It is provided so that new translators can call it to get default KeyCode-to-KeySym translations and so that the default translator can be reinstalled.

The **XtTranslateKeycode** function passes the specified arguments directly to the currently registered KeyCode to KeySym translator.

The **XtRegisterCaseConverter** registers the specified case converter. The start and stop arguments provide the inclusive range of KeySyms for which this converter is to be called. The new converter overrides any previous converters for KeySyms in that range. No interface exists to remove converters; you need to register an identity converter. When a new converter is registered, the Intrinsics refreshes the keyboard state if necessary. The default converter understands case conversion for all KeySyms defined in the core protocol.

The **XtConvertCase** function calls the appropriate converter and returns the results. A user-supplied **XtKeyProc** may need to use this function.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtSetKeyboardFocus – focus events on a child widget

SYNTAX

XtSetKeyboardFocus(*subtree, descendant*)
Widget *subtree, descendant*;

ARGUMENTS

descendant Specifies either the widget in the subtree structure which is to receive the keyboard event, or **None**. Note that it is not an error to specify **None** when no input focus was previously set.

w Specifies the widget for which the keyboard focus is to be set.

DESCRIPTION

If a future **KeyPress** or **KeyRelease** event occurs within the specified subtree, **XtSetKeyboardFocus** causes **XtDispatchEvent** to remap and send the event to the specified descendant widget.

When there is no modal cascade, keyboard events can occur within a widget W in one of three ways:

- W has the X input focus.
- W has the keyboard focus of one of its ancestors, and the event occurs within the ancestor or one of the ancestor's descendants.
- No ancestor of W has a descendant within the keyboard focus, and the pointer is within W.

When there is a modal cascade, a widget W receives keyboard events if an ancestor of W is in the active subset of the modal cascade and one or more of the previous conditions is **True**.

When subtree or one of its descendants acquires the X input focus or the pointer moves into the subtree such that keyboard events would now be delivered to subtree, a **FocusIn** event is generated for the descendant if **FocusNotify** events have been selected by the descendant. Similarly, when W loses the X input focus or the keyboard focus for one of its ancestors, a **FocusOut** event is generated for descendant if **FocusNotify** events have been selected by the descendant.

SEE ALSO

XtCallAcceptFocus(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtSetSensitive, XtIsSensitive – set and check a widget's sensitivity state

SYNTAX

void XtSetSensitive(*w, sensitive*)
Widget *w*;
Boolean *sensitive*;

Boolean XtIsSensitive(*w*)
Widget *w*;

ARGUMENTS

sensitive Specifies a Boolean value that indicates whether the widget should receive keyboard and pointer events.

w Specifies the widget.

DESCRIPTION

The **XtSetSensitive** function first calls **XtSetValues** on the current widget with an argument list specifying that the sensitive field should change to the new value. It then recursively propagates the new value down the managed children tree by calling **XtSetValues** on each child to set the ancestor_sensitive to the new value if the new values for sensitive and the child's ancestor_sensitive are not the same.

XtSetSensitive calls **XtSetValues** to change sensitive and ancestor_sensitive. Therefore, when one of these changes, the widget's set_values procedure should take whatever display actions are needed (for example, greying out or stippling the widget).

XtSetSensitive maintains the invariant that if parent has either sensitive or ancestor_sensitive **False**, then all children have ancestor_sensitive **False**.

The **XtIsSensitive** function returns **True** or **False** to indicate whether or not user input events are being dispatched. If both core.sensitive and core.ancestor_sensitive are **True**, **XtIsSensitive** returns **True**; otherwise, it returns **False**.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtSetValues, XtSetSubvalues, XtGetValues, XtGetSubvalues – obtain and set widget resources

SYNTAX

void XtSetValues(*w, args, num_args*)
Widget *w*;
ArgList *args*;
Cardinal *num_args*;

void XtSetSubvalues(*base, resources, num_resources, args, num_args*)
XtPointer *base*;
XtResourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

void XtGetValues(*w, args, num_args*)
Widget *w*;
ArgList *args*;
Cardinal *num_args*;

void XtGetSubvalues(*base, resources, num_resources, args, num_args*)
XtPointer *base*;
XtResourceList *resources*;
Cardinal *num_resources*;
ArgList *args*;
Cardinal *num_args*;

ARGUMENTS

args	Specifies the argument list of name/address pairs that contain the resource name and either the address into which the resource value is to be stored or their new values.
base	Specifies the base address of the subpart data structure where the resources should be retrieved or written.
num_args	Specifies the number of arguments in the argument list.
resources	Specifies the nonwidget resource list or values.
num_resources	Specifies the number of resources in the resource list.
w	Specifies the widget.

DESCRIPTION

The **XtSetValues** function starts with the resources specified for the **Core** widget fields and proceeds down the subclass chain to the widget. At each stage, it writes the new value (if specified by one of the arguments) or the existing value (if no new value is specified) to a new widget data record. **XtSetValues** then calls the set_values procedures for the widget in superclass-to-subclass order. If the widget has any non-NULL set_values_hook fields, these are called immediately after the corresponding set_values procedure. This procedure permits subclasses to set nonwidget data for **XtSetValues**.

If the widget's parent is a subclass of **constraintWidgetClass**, **XtSetValues** also updates the widget's constraints. It starts with the constraint resources specified for **constraintWidgetClass** and proceeds down the subclass chain to the parent's class. At each stage, it writes the new value or the existing value to a new constraint record. It then calls the constraint set_values procedures from **constraintWidgetClass** down to the parent's class. The constraint set_values procedures are called with widget arguments, as for all set_values procedures, not just the constraint record arguments, so that they can make adjustments to the desired values based on full information about the widget.

XtSetValues determines if a geometry request is needed by comparing the current widget to the new widget. If any geometry changes are required, it makes the request, and the geometry manager returns **XtGeometryYes**, **XtGeometryAlmost**, or **XtGeometryNo**. If **XtGeometryYes**, **XtSetValues** calls the widget's resize procedure. If **XtGeometryNo**, **XtSetValues** resets the geometry fields to their original values. If **XtGeometryAlmost**, **XtSetValues** calls the set_values_almost procedure, which determines what should be done and writes new values for the geometry fields into the new widget. **XtSetValues** then repeats this process, deciding once more whether the geometry manager should be called.

Finally, if any of the set_values procedures returned **True**, **XtSetValues** causes the widget's expose procedure to be invoked by calling the Xlib **XClearArea** function on the widget's window.

The **XtSetSubvalues** function stores resources into the structure identified by base.

The **XtGetValues** function starts with the resources specified for the core widget fields and proceeds down the subclass chain to the widget. The value field of a passed argument list should contain the address into which to store the corresponding resource value. It is the caller's responsibility to allocate and deallocate this storage according to the size of the resource representation type used within the widget.

If the widget's parent is a subclass of **constraintWidgetClass**, **XtGetValues** then fetches the values for any constraint resources requested. It starts with the constraint resources specified for **constraintWidgetClass** and proceeds down to the subclass chain to the parent's constraint resources. If the argument list contains a resource name that is not found in any of the resource lists searched, the value at the corresponding address is not modified. Finally, if the get_values_hook procedures are non-NULL, they are called in superclass-to-subclass order after all the resource values have been fetched by **XtGetValues**. This permits a subclass to provide nonwidget resource data to **XtGetValues**.

The **XtGetSubvalues** function obtains resource values from the structure identified by base.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtStringConversionWarning – issue a conversion warning message

SYNTAX

void XtStringConversionWarning(*src*, *dst_type*)
String *src*, *dst_type*;

ARGUMENTS

src	Specifies the string that could not be converted.
dst_type	Specifies the name of the type to which the string could not be converted.

DESCRIPTION

The **XtStringConversionWarning** function issues a warning message with name "conversionError", type "string", class "XtToolkitError, and the default message string "Cannot convert "*src*" to type *dst_type*".

SEE ALSO

XtAppAddConverter(3), XtAppErrorMsg(3t), XtConvert(3)
X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

NAME

XtTranslateCoordinates – translate widget coordinates

SYNTAX

void XtTranslateCoords(*w*, *x*, *y*, *rootx_return*, *rooty_return*)
Widget *w*;
Position *x*, *y*;
Position **rootx_return*, **rooty_return*;

ARGUMENTS

rootx_return	
rooty_return	Returns the root-relative x and y coordinates.
x	
y	Specify the widget-relative x and y coordinates.
w	Specifies the widget.

DESCRIPTION

While **XtTranslateCoords** is similar to the Xlib **XTranslateCoordinates** function, it does not generate a server request because all the required information already is in the widget's data structures.

SEE ALSO

X Toolkit Intrinsics - C Language Interface
Xlib - C Language X Interface

NOTE

CXwindows is an optional product; for more information, contact your CONVEX sales representative.

CONVEX CXwindows Programmer's Reference
710-004530-002, Second Edition

Reader's Forum

Please use this form to submit comments or questions concerning the clarity and service of this manual. Constructive critical comments are most welcome and help us continue in our efforts to generate quality customer documentation. Please list the page number for questions or comments.

_____

_____

_____

_____

_____

_____

_____

_____

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

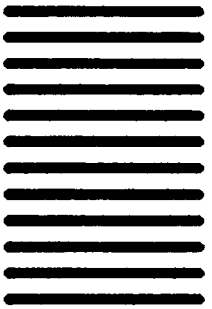
Location	Phone Number
From all locations in continental U.S.	1(800)952-0379
From locations in Alaska, Hawaii, & Canada	1(214)497-4379
From all other locations	Contact nearest CONVEX office

Direct mail orders to:

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851 USA

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

CONVEX Computer Corporation
Customer Service
PO Box 833851
Richardson TX 75083-3851
USA

(Fold Here Second)

(Tape or Staple)