X Window System Reference Manual
Version 11

Integrated Solutions
1140 Ringwood Court
San Jose, California 95131
(408) 943-1902

490242 Rev. A

September 1988

PREFACE

The X Window System Reference Manual provides manual pages for each Xlib function and manual pages for each X Window command. The manual pages are broken into two sections

- **Section 1:** X Window manual pages, and
- **Section 3X:** Xlib manual pages.

Within each section, manual pages are presented in alphabetical order.

A permuted index is provided to help locate the correct manual page. To use the permuted index, simply scan down the middle of the page looking for a keyword of interest on the right side of the blank gutter. When you find the keyword you want, read the brief description of the command that makes up the entry. If you believe you've found the command you were looking for, the far right-hand column provides the relevant command page.

Each manual page is broken down into several parts, each part containing information relevant to using the command. An explanation of the manual page sub-headings follows:

- **NAME:** Under this heading appear the commands related to a particular manual page. The first command name is the command after which the manual page is named. Following the command names (after the "-") is a brief description of the commands.
- **SYNTAX:** Under this heading appear variations of command syntax. Samples of syntax argument order are provided.
- **OPTIONS:** Under this heading appear definitions of syntax options. In some cases, specific instructions for user input are provided.
- **DESCRIPTION:** Under this heading appear descriptions of all commands called out under the NAME heading. Any errors that can be generated from use of a specific command are provided here also.
- **BUGS:** Under this heading appear any known bugs associated with the command.
- **DIAGNOSTICS:** Under this heading appear descriptions of error messages introduced under the DESCRIPTION heading.
- **SEE ALSO:** Under this heading appear related manual pages. Each manual page also refers you to the *Xlib - C Language X Interface, Protocol Version 11* documentation. This documentation is provided on-line and in the X Window System Core documentation package.

Xlib Description

The Xlib or X library is the lowest level of programming interface to the X Window System. It is powerful enough to write effective applications without additional programming tools, and is necessary even with the toolkit. The X Toolkit simplifies the process considerably, providing a number of *widgets* (an object providing a user-interface abstraction; for example, a scroll bar) that implement menus, command buttons, and other common features of the user interface. Xlib and the X Toolkit should be used together in virtually all applications.

Briefly, the X Toolkit is a user-interface library package that is layered on top of the X Window System. The X Toolkit provides tools that simplify the design of application user interfaces in the X programming environment. It assists application programmers by providing a commonly used set of underlying user-interface functions that enable you to manage widgets, standard operations, toolkit initialization, memory, input and events, widget geometry, input focus, selections, resources and resource conversion, translation of events, graphics contexts, pixmaps, and errors. (For more information on the X Toolkit, refer to the *X*

Toolkit Library - C Language Interface in the core documentation package).

Xlib is a C subroutine library that application programs (clients) use to interface with the window system by means of a stream connection. Although a client usually runs on the same machine as the X server it is talking to, this need not be the case.

Most of the functions in Xlib just add requests to an output buffer. These requests later execute asynchronously on the X server, often referred to as display server. Functions that return values of information stored in the server do not return (that is, they "block") until an explicit reply is received or an error occurs. If a nonblocking call results in an error, the error will generally not be reported by a call to an optional error handler until some later blocking call is made.

Many Xlib functions will return an integer resource ID, which allows you to refer to objects stored on the X server. These can be of type Window, Font, Pixmap, Cursor, and GContext, as defined in a file. These resources are created by user requests, and destroyed (or freed) by user requests or when connections are closed. Most of these resources are potentially sharable between applications, and in fact, windows are manipulated explicitly by window manager programs. Fonts and cursors are typically shared automatically since the X server treats fonts specially, loading and unloading font storage as needed. GContexts should not be shared between applications.

As previously stated, Xlib consists of functions. There are several different sets of functions within Xlib. Those sets of functions are briefly described here (for more information on the function sets, refer to the *Xlib - C Language X Interface, Protocol Version 11* on-line or core documentation).

- **Xlib window functions** allow you to: create windows; destroy windows; map windows; unmap windows; configure windows; change the stacking order; change the window attributes; and translate window coordinates.
- **Xlib window information functions** allow you to: obtain information about a window; manipulate property lists; obtain and change window properties; and manipulate window selection.
- **Xlib graphics resource functions** allow you to: manipulate the colormap; manipulate pixmaps; manipulate graphics context/state; and use GC convenience routines.
- **Xlib graphics functions** allow you to: clear and copy areas; draw points, lines, rectangles, and arcs; manipulate fonts; draw text characters; transfer images between clients and server; and manipulate cursors.
- **Xlib window manager functions** allow you to: change the parent of a window; control the lifetime of a window; manipulate the colormap; manipulate the pointer; manipulate the keyboard; grab the server; control processing; manipulate the keyboard encoding; manipulate the screen saver; and control host access.
- **Xlib events and event-handling functions** allow you to: select events; handle the output buffer and the event queue; select events from the event queue; send and get events; and handle error events.
- **Xlib predefined property functions** allow you to: communicate with window managers; and manipulate standard colormaps.
- **Xlib application utility functions** allow you to: handle keyboard events; obtain the X environment defaults; parse the window geometry; parse the hardware colors; generate regions; manipulate regions; use the cut and paste buffers; determine the appropriate visual; manipulate images; manipulate bitmaps; use the resource manager; and use the context manager.

Naming Conventions within Xlib

Throughout Xlib, a number of conventions for naming and syntax of the Xlib functions have been followed. These conventions are intended to make the syntax of the functions more predictable, given that you remember what information the routine may require.

The major naming conventions are:

- The library uses mixed case for external symbols, and leaves lower case for variables and all upper case for user macros. This distinction helps to better differentiate the X symbols from the user symbols.
- All Xlib functions begin with a capital X.
- The beginnings of all procedure names and symbols are capitalized.
- All user-visible data structures begin with a capital X. More generally, anything that a user might dereference begins with a capital X.
- Macros and other symbols do not begin with a capital X. To distinguish them from all user symbols, each word in the macro is capitalized.
- All elements of or variables in a data structure are in lower case. Compound words, where needed, are constructed with underscores (_).
- Global symbols in this manual are printed in this font. These can be either procedure names, symbols defined in include files, or structure names.

There are also conventions for argument order. These conventions are:

- The display argument, where used, is always first in the argument list.
- All resource objects, where used, occur at the beginning of the argument list, immediately after the display variable.
- When a graphics context is present together with another type of resource (most commonly, a drawable), the graphics context occurs in the argument list after the other resource. Drawables outrank all other resources.
- Source arguments always precede the destination arguments in the argument list.
- The x argument always precedes the y argument in the argument list. The width argument always precedes the height argument in the argument list.
- Where the x, y, width, and height arguments are used together, the x and y arguments always precede the width and height arguments.
- Where an array occurs in an argument list accompanied with a count (number of elements in the array), the array always precedes the count in the argument list.
- Where a mask is accompanied with a structure, the mask always precedes the pointer to the structure in the argument list.
- Arguments, user-supplied variables, are printed in *italics*.
- To eliminate any ambiguity between those arguments that you pass and those that a function returns to you, the explanations for all arguments that you pass start with the word "specifies." By contrast, the explanations for all arguments that are returned to you, start with the word "returns."

PERMUTED INDEX

TABLE OF CONTENTS

1. X Window System Commands and Programs

3X. Xlib Routines

TABLE OF CONTENTS

1. X Window System Commands and Programs

NAME

X - a portable, network-transparent window system

SYNOPSIS

X is a network-transparent window system developed at MIT which runs under a wide variety of operating systems. The standard distribution from MIT works on Ultrix-32 Version 1.2 (and higher), 4.3BSD Unix, SunOS 3.2 (and higher), HP-UX 6.01, and DOMAIN/IX 9.7. In addition, many vendors support the X Window System under other operating systems.

THE OFFICIAL NAMES

The official names of the software described herein are:

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

Note that the phrases X.11, X-11, X Windows or any permutation thereof, are explicity excluded from this list and should not be used to describe the X Window System (window system should be thought of as one word).

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 functions, see the *Xlib - C Language X Interface* manual, the *X Window System Protocol* specification, and various toolkit documents.

When you first log in on a display running X, you are usually using the **xterm**(1) terminal emulator program. You need not learn anything extra to use a display running X as a terminal beyond moving the mouse cursor into the login window to log in normally.

The core X protocol provides mechanism, not policy. Windows are manipulated (including moving, resizing and iconifying) not by the server itself, but by a separate program called a "window manager" of your choosing. This program is simply another client and requires no special privileges. If you don't like the ones that are supplied (see **uwm**(1) and **wm**(1)), you can write your own.

The number of programs that use X is growing rapidly. Of particular interest are: a terminal emulator (**xterm**(1)), window managers (**wm**(1) and **uwm**(1)), a mailer reader (**xmh**(1)), a bitmap editor (**bitmap**(1)), an access control program (**xhost**(1)), user preference setting programs (**xset**(1), **xsetroot**(1), and **xmodmap**(1)), a load monitor (**xload**(1)), clock (**xclock**(1)), a font displayer (**xfd**(1)), a protocol translator for running X10 programs (**x10tox11**(1)), and various demos (**ico**(1), **muncher**(1), **puzzle**(1), etc.).

DISPLAY SPECIFICATION

When you first log in, the environment variable DISPLAY is set to a string specifying the name of the machine on which the server is running, a number indicating which of possibly several servers to use, and possibly a number indicating the default screen of the server (usually this is omitted and defaults to 0). By convention, servers on a particular machine are numbered starting with zero. The format of the DISPLAY string depends on the type of communications channel used to contact the server.

The following connection protocols are supported:

TCP/IP

DISPLAY should be set to *host:dpy.screen* where *host* is the symbolic name of the machine (e.g., expo), *dpy* is the number of the display (usually 0), and *screen* is the number of the screen. The *screen* and preceding period are optional, with the default value being zero (0). Full Internet domain names (e.g., *expo.lcs.mit.edu*) are allowed for the host name.

Unix domain

DISPLAY should be set to unix:*dpy.screen*, where *dpy* is the display number and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

DECnet

DISPLAY should be set to *nodename::dpy.screen* where *nodename* is the symbolic name of the machine, *dpy* is the display number, and *screen* is the screen number; *screen* and the preceding period are optional, with the default value being zero (0).

Most programs accept a command line argument of the form "-display *display*" that can be used to override the DISPLAY environment variable.

GEOMETRY SPECIFICATION

One of the advantages of using window systems over 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, most applications accept a command line argument that is treated as the prefered size and location for this particular application's window.

This argument, usually specified as "-geometry WxH+X+Y," indicates that the window should have a width of W and height of H (usually measured in pixels or characters, depending on the application), and the upper-left corner X pixels to the right and Y pixels below the upper-left corner of the screen (origin (0,0)). "WxH" can be omitted to obtain the default application size, or "+X+Y" can be omitted to obtain the default application position (which is usually then left up to the window manager or user to choose). The X and Y values may be negative to position the window off the screen. In addition, if minus signs are used instead of plus signs (e.g., WxH-X-Y), then (X,Y) represents the location of the lower-right hand corner of the window relative to the lower-right hand corner of the screen.

By combining plus and minus signs, the window may be place relative to any of the four corners of the screen. For example:

555x333+11+22	This will request a window 555 pixels wide and 333 pixels tall, with the upper-left corner located at (11,22).
300x200-0+0	This will request a window measuring 300 by 200 pixels in the upper-right hand corner of the screen.
48x48--5--10	This will request a window measuring 48 by 48 pixels whose lower-right hand corner is 5 pixel off the right edge and the screen and 10 pixels off the bottom edge.

OPTIONS

Most X programs attempt to use a common set of names for their command line arguments. The X Toolkit automatically handles the following arguments:

–bg *color*, **–background** *color*	Specifies the color to use for the window background.
–bd *color*, **–bordercolor** *color*	Specifies the color to use for the window border.
–bw *number*, **–borderwidth** *number*	Specifies the width in pixels of the window border.
–display *display*	Specifies the name of the X server to use.
–fg *color*, **–foreground** *color*	Specifies the color to use for text or graphics.

–fn *font*, **-font** *font* — Specifies the font to use for displaying text.

–geometry *geometry* — Specifies the initial size and location of the window.

–iconic — Indicates that application should start out in an iconic state. Note that how this state is represented is controlled by the window manager that you are running.

–name — 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 — 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 — Indicates that the program should not simulate reverse video. This is used to override any defaults since reverse video doesn't always work properly.

–synchronous — 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* — 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.

–xrm *resourcestring* — 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 *"name*subname*subsubname...: value"* (see the *Xlib* manual section *Using the Resource Manager* for more details) that are loaded into a client when it starts up. The *Xlib* routine *XGetDefault(3X)* and the resource utilities within the X Toolkit 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**(1) program.

application-specific directory — Any application- or machine-specific resources can be stored in the class resource files located in the XAPPLOADDIR directory (this is a configuration parameter that is /usr/lib/X11/app-defaults in the standard distribution).

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, the X Toolkit looks for a file named .Xdefaults-*hostname*, where *hostname* is the name of the host where the application is executing.

–xrm *resourcestring* Applications that use the X Toolkit 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 "instance" resources can be set all at once. By convention, the instance name of a resource begins with a lowercase letter and class name with an uppercase letter. Multiple word resources are concatentated with the first letter of the succeeding words capitalized. Applications written with the X Toolkit will have at least the following resources:

background (class **Background**) Specifies the color to use for the window background.

borderWidth (class **BorderWidth**) Specifies the width in pixels of the window border.

borderColor (class **BorderColor**) Specifies the color to use for the window border.

Most X Toolkit applications 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.

When a named resource is unavailable (for example, a color named chartrusse or a font named teeneyweeney), normally no error message will be printed; whether or not useful results ensue is dependent on the particular application. If you wish to see error messages (for example, if an application is failing for an unknown reason), you may specify the value "on" for the resource named "StringConversionWarnings." If you want such warnings for all applications, specify "*StringConversionWarnings:on" to the resource manager. If you want warnings only for a single application named "zowie", specify "zowie*StringConversionWarnings:on" to the resource manager.

DIAGNOSTICS

The default error handler uses the Resource Manager to build diagnostic messages when error conditions arise. The default error database is stored in the file *XErrorDB* in the directory specified by the LIBDIR configuration parameter (*/usr/lib/X11* in the standard distribution). If this file is not installed, error messages will tend to be somewhat cryptic.

SEE ALSO

xterm(1), **bitmap**(1), **ico**(1), **muncher**(1), **plaid**(1), **puzzle**(1), **resize**(1), **uwm**(1), **wm**(1), **x10tox11**(1), **xbiff**(1), **xcalc**(1), **xclock**(1), **xedit**(1), **xfd**(1), **xhost**(1), **xinit**(1), **xload**(1), **xlogo**(1), **xlsfonts**(1), **xmh**(1), **xmodmap**(1), **xpr**(1), **xprkbd**(1), **xprop**(1), **xrdb**(1), **xrefresh**(1), **xset**(1), **xsetroot**(1), **xwd**(1), **xwininfo**(1), **xwud**(1), **Xserver**(1), **biff**(1), **init(8)**, **ttys(5)**, *Xlib – C Language X Interface, X Toolkit Intrinsics - C Language X Interface*

NAME

X - X Window System server

SYNOPSIS

X displaynumber [-option ...] ttyname

DESCRIPTION

X is the window system server. On operating systems derived from 4.3BSD, it is run automatically by **init**(8), otherwise it is started from the **xinit**(1) program. The *displaynumber* argument is used by clients in their DISPLAY environment variables to indicate which server to contact (large machines may have several displays attached). This number is usually in the range of 0-6 and is also used in determining the names of various startup files. The *ttyname* argument is passed in by *init* and isn't used.

The executable that is invoked when **X** is run is actually one of a collection of programs that depend on the hardware that is installed on the machine. Any additional features are described in the documentation for that server.

The sample server has support for the following protocols:

TCP/IP	The server listens on port htons(6000+N), where N is the display number.
Unix Domain	The file name for the socket is X_UNIX_PATH* where X_UNIX_PATH is a configuration parameter (*/tmp/X11-unix/X* in the standard release) and "*" is the display number.
DECnet	The server responds to connections to object "X*", where "*" is the display number.

When the sample server starts up, it takes over the display. If you are running on a workstation whose console is the display, you cannot log into the console while the server is running.

OPTIONS

The following options can be given on the command line to any X server, usually when it is started by **init**(1) using information stored in the file */etc/ttys*. (see **ttys**(5) for details):

–a *number*	Sets pointer acceleration (i.e., the ratio of how much is reported to how much you actually moved the pointer).
–c	Turns off key-click.
c *volume*	Sets key-click volume (allowable range: 0-8).
–f *volume*	Sets feep (bell) volume (allowable range: 0-7).
–logo	Turns on the X Window System logo display in the screen-saver. There is currently no way to change this from a client.
nologo	Turns off the X Window System logo display in the screen-saver. There is currently no way to change this from a client.
–p *minutes*	Sets screen-saver pattern cycle time in minutes.
–r	Turns off auto-repeat.
r	Turns on auto-repeat.
–s *minutes*	Sets screen-saver timeout time in minutes.
–t *numbers*	Sets pointer acceleration threshold in pixels (i.e., after how many pixels pointer acceleration should take effect).
–to *seconds*	Sets default screensaver timeout in seconds.
v	Sets video-on screen-saver preference.
–v	Sets video-off screen-saver preference.

–co *filename*	Sets the name of the RGB color database.
–help	Prints a usage message.
–fp *fontPath*	Sets the search path for fonts.
–fc *cursorFont*	Sets the default cursor font.
–fn *font*	Sets the default font.

Specific implementations may have other command line options.

RUNNING FROM INIT

On operating systems such as 4.3BSD and Ultrix, the server and your login window are normally started automatically by **init**(8).

By convention, the pseudoterminal with the highest minor device number (e.g., */dev/ttyqf* and */dev/ptyqf*) is renamed for the lowest display number (e.g., */dev/ttyv0* and */dev/ptyv0*). Machines that have more than one display can repeat this process using *ttyqe* for *ttyv1*, and so on.

With this done, you can set up */etc/ttys* to run **X** and **xterm** by adding the following entry (the \'s is to make the lines fit on this manual page, the real entries have to occupy just one line):

```
ttyv0 "/etc/xterm -L -geometry -1+1 -display :0" xterm on \
        secure window="/etc/X :0 -c -l"
```

on the client machine.

Important note: some versions of *init* have relatively small program name buffer sizes, so you may find that you can't list many **xterm** options. In addition, some *init*'s will treat the sharp signs that are used in specifying colors (such as for window backgrounds) as comments for the whole line. If you run into problems such as this you may want to write a small program that just exec's **xterm** with the proper arguments and have *init* run that instead.

If all else fails, set the display up to be a dumb terminal and use the **xinit** program after logging in.

SECURITY

X uses an access control list for deciding whether or not to accept a connection from a given cleint. This list initially consists of the machine on which the server is running, and any hosts listed in the file */etc/X*.hosts* (where * is the display number). This file should contain one line per host name, with no white space. If a name ends in "::", it is assumed to be a DECnet host, otherwise it is interpreted as an Internet host.

The user can manipulate a dynamic form of this list in the server using the **xhost**(1) program from the same machine as the server.

Unlike some window systems, **X** does not have any notion of window operation permissions or place any restrictions on what a client can do; if a program can connect to a display, it has full run of the screen. There is support for using authentication services on connection startup beyond the simple host name check, but it is not used in the standard distribution.

SIGNALS

X will catch the SIGHUP signal sent by **init**(8) after the initial process (usually the login **xterm**(1)) started on the display terminates. This signal causes all connections to be closed (thereby "disowning" the terminal), all resources to be freed, and all defaults restored.

DIAGNOSTICS

Too numerous to list them all. If run from **init**(8), errors are logged in the file */usr/adm/X*msgs*.

FILES

/etc/X*.hosts	Initial access control list.
/usr/lib/X11/fonts	Font directory.

/usr/lib/X11/rgb.txt	Color database.
/tmp/.X11-unix/X*	Unix domain socket.
/usr/adm/X*msgs	Error log file.

SEE ALSO

X(1), **xinit**(1), **xterm**(1), **xwm**(1), **xhost**(1), **xset**(1), **xsetroot**(1), **ttys**(5), **init**(8), *X Window System Protocol, Definition of the Porting Layer for the X v11 Sample Server, Strategies for Porting the X v11 Sample Server.*

BUGS

The option syntax is inconsistent with itself and **xset**(1).

The acceleration option should take a numerator and a denominator like the protocol.

If **X** dies before its clients, new clients won't be able to connect until all existing connections have their TCP TIME_WAIT timers expire.

The color database is missing a large number of colors. However, there doesn't seem to be a better one available that can generate RGB values.

NAME

bitmap – bitmap editor for X

SYNTAX

bitmap [*–options*] *filename* [*WIDTHxHEIGHT*]

DESCRIPTION

Bitmap lets you interactively create bitmaps, or edit previously created bitmaps. A bitmap is simply a rectangular array of 0 and 1 bits. The X Window System uses bitmaps in defining clipping regions, cursor shapes, icon shapes, and tile and stipple patterns.

When you run **bitmap,** you are given a magnified version of the bitmap, in which each bit is shown as a large square, as if it were a piece of graph paper. The pointer can be used to set, clear, or invert individual squares, and to invoke commands to set, clear or invert larger rectangular areas of the bitmap. Other commands may be used to move or copy rectangular areas from one part of the bitmap to another, and to define a hot spot (a special single point on the bitmap, which is useful when the bitmap is used as an X cursor).

The output of the **bitmap** program is a small C code fragment. By #include'ing such a program fragment in your program, you can easily declare the size and contents of cursors, icons, and other bitmaps that your program creates to deal with the X Window System.

OPTIONS

–help	Causes a brief description of the allowable options and parameters to be printed.
–display *display*	Specifies the server to be used. See X(1) for details.
–geometry *geometry*	Specifies the placement and size of the bitmap window on the screen. See X(1) for details.
–nodashed	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.
–bw *number*	Specifies the border width in pixels of the main window.
–fn *font*	Specifies the font to be used in the buttons.
–fg *color*	Specifies the color to be used for the foreground.
–bg *color*	Specifies the color to be used for the background.
–hl *color*	Specifies the color to be used for highlighting.
–bd *color*	Specifies the color to be used for the window border.
–ms *color*	Specifies the color to be used for the pointer (mouse).

When **bitmap** starts, it first tries to read the specified file (see FILE FORMAT). If the file already exists, it creates a window containing a grid of the appropriate dimensions.

If the file does not exist, **bitmap** creates a window for a bitmap of the size specified by *WIDTHxHEIGHT* (e.g., 7x9, 13x21). The bitmap starts out empty. If *WIDTHxHEIGHT* is not specified either on the command line or in the Dimensions X Default, 16x16 is assumed.

The window that **bitmap** creates has four parts. The largest section is the checkerboard grid, which is a magnified version of the bitmap you are editing. At the upper-right is a set of commands that you can invoke with any pointer button. Below the commands is an actual size picture of the bitmap you are editing; below that is an inverted version of the same bitmap. Each time you alter the image in the grid, the change is reflected in the actual-size versions of the bitmap.

If you use a window manager to make the **bitmap** window larger or smaller, the grid squares automatically get larger or smaller as well.

COMMANDS

(Note for users of color displays: In all of the following, white means the background color, and black means the foreground color.)

When the cursor is in the checkerboard region, each pointer button has a different effect upon the single square that the cursor is over:

Button 1 (The left mouse button.) Sets the indicated square.

Button 2 (The middle mouse button.) Inverts the indicated square.

Button 3 (The right mouse button.) Clears the indicated square.

The various commands are invoked by pressing any pointer button in the corresponding command box:

Clear All Clears all squares in the bitmap. *Caution:* This command is irreversible, so invoke it with care.

Set All Sets all squares in the bitmap. *Caution:* This command is irreversible, so invoke it with care.

Invert All Inverts all squares in the bitmap.

Clear Area Clears a rectangular area of the bitmap. After you click over this command, the cursor turns into an upper-left corner. Press any pointer button over the upper-left corner of the area you want to clear, hold the button down while moving the pointer to the lower-right corner of the area you want to clear, and then release the button.

While you are holding down the button, the selected area will be covered with X's, and the cursor will change to a lower-right corner. If you now wish to abort the command without clearing an area, either press another pointer button, move the cursor outside the grid, or move the cursor to the left of or above the upper-left corner.

Set Area Sets a rectangular area of the bitmap. It works the same way as the **Clear Area** command.

Invert Area Inverts a rectangular area of the bitmap. It works the same way as the **Clear Area** command.

Copy Area Copies a rectangular area from one part of the grid to another. First, you select the rectangle to be copied, in the manner described above under **Clear Area** above. Then, the cursor changes to an upper-left corner. When you press a pointer button, a destination rectangle overlays the grid; moving the pointer while holding down the button moves this destination rectangle. The copy occurs when you release the button. To cancel the copy, move the pointer outside the grid and then release the button.

Move Area Works identically to **Copy Area**, except that it clears the source rectangle after copying to the destination.

Line Draws a line between two points.

Overlay Area
Works identically to **Copy Area**, except that it does a binary OR of the source rectangle with the destination.

Circle Draws a circle specifying the center and a radius.

Filled Circle Draw a filled circle given the center and radius of the circle.

Set HotSpot Designates a point on the bitmap as the hot spot. If a program is using your bitmap as a cursor, the hot spot indicates which point on the bitmap is the actual location of the cursor. For instance, if your cursor is an arrow, the hot spot should be the tip of the arrow; if your cursor is a cross, the hot spot should be where the perpendicular lines intersect.

Clear HotSpot
Removes any hot spot that was defined on this bitmap.

Write Output
Writes the current bitmap value to the file specified in the original command line. If the file already exists, the original file is first renamed to **filename~** (in the manner of **emacs**(1) and other text editors).

If either the renaming or the writing cause an error (e.g., "Permission denied"), a dialog window appears, asking if you want to write the file */tmp/filename* instead. If you say yes, all future Write Output commands write to */tmp/filename* as well. See below for the format of the output file.

Quit Exits the **bitmap** program. If you have edited the bitmap and have not invoked **Write Output**, or you have edited it since the last time you invoked **Write Output**, a dialog window appears, asking if you want to save changes before quitting. "Yes" does a "Write Output" before exiting; "No" just exits, losing the edits; "Cancel" means you decided not to quit after all.

FILE FORMAT

Bitmap reads and writes files in the following format, which is suitable for #include'ing in a C program:

```
#define name_width 9
#define name_height 13
#define name_x_hot 4
#define name_y_hot 6
static char name_bits[] = {
  0x10, 0x00, 0x38, 0x00, 0x7c, 0x00, 0x10, 0x00, 0x10, 0x00, 0x10, 0x00,
  0xff, 0x01, 0x10, 0x00, 0x10, 0x00, 0x10, 0x00, 0x7c, 0x00, 0x38, 0x00,
  0x10, 0x00};
```

The variables ending with *_x_hot* and *_y_hot* are optional; they are present only if a hot spot has been

defined for this bitmap. The other variables must be present.

The *name* portion of the five variables are derived from the name of the file that you specified on the original command line by
(1) deleting the directory path (all characters up to and including the last "/," if one is present).
(2) deleting the extension (the first ".", if one is present, and all characters beyond it).

For example, invoking **bitmap** with filename */usr/include/bitmaps/cross.bitmap* produces a file with variable names *cross_width*, *cross_height*, and *cross_bits* (and *cross_x_hot* and *cross_y_hot* if a hot spot is defined).

It's easy to define a bitmap or cursor in an X program by simply #include'ing a bitmap file and referring to its variables. For instance, to use a cursor defined in the files *this.cursor* and *this_mask.cursor*, one simply writes

```
#include "this.cursor"
#include "this_mask.cursor"
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);
```

where *foreground* and *background* are **XColor**(3X) values.

An X program can also read a bitmap file at runtime by using the function **XReadBitmapFile**(3X).

The bits are in **XYBitmap**(3X) format, with *bitmap_unit = bitmap_pad = 8*, and *byte_order = bitmap_bit_order = LSBFirst* (least significant bit and byte are leftmost).

For backward compatibility with X10, **bitmap** can also read in a file where the bits array is declared as *static short foo_bits[]* and consists of an array of 16-bit hex constants. This is interpreted as a **XYBitmap(3X)** with *bitmap_unit = bitmap_pad = 16, byte_order bitmap_bit_order = LSBFirst.* If you modify the bitmap after reading in such a file, **bitmap** will always write the file back out in standard X11 format.

X DEFAULTS

The **bitmap** program uses the routine **XGetDefault**(3X) to read defaults, so its resource names are all capitalized.

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."
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 (clearing). 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.

ENVIRONMENT

DISPLAY	The default host and display number.
XENVIRONMENT	The name of the default file to use.

SEE ALSO

X(1), emacs(1), XReadBitmapFile(3), XGetDefault(3X), *Xlib - C Language X Interface*, particularly the section "Manipulating Bitmaps."

DIAGNOSTICS

The following messages may be printed to the standard error output. Any of these conditions aborts **bitmap** before it can create its window.

"bitmap: could not connect to X server on *host:display*"

Either the display given on the command line or the DISPLAY environment variable has an invalid host name or display number, or the host is down, or the host is unreachable, or the host is not running an X server, or the host is refusing connections.

"bitmap: no file name specified"

You invoked **bitmap** with no command line arguments. You must give a file name as the first argument.

"bitmap: could not open file *filename* for reading -- *message*"

The specified file exists but cannot be read, for the reason given in <message> (e.g., permission denied).

"bitmap: invalid dimensions *string*"
"bitmap: dimensions must be positive"

The second command line argument was not a valid dimension specification.

"bitmap: Bitmap file invalid"

The input file is not in the correct format; the program gave up when trying to read the specified data.

The following messages may be printed after **bitmap** creates its window:

"bitmap: Unrecognized variable *name* in file *filename*"

Bitmap encountered a variable ending in something other than *_x_hot*, *_y_hot*, *_width*, or *_height* while parsing the input file. It will ignore this variable and continue parsing the file.

"bitmap: XError: *message*"
"bitmap: XIOError"

A protocol error occurred. Something is wrong with either the X server or the X library which the program was compiled with. Possibly they are incompatible. If the server is not on the local host, maybe the connection broke.

BUGS

The old command line arguments aren't consistent with other X programs.

The foreground, background, and highlight colors are ignored unless new values for all three are specified.

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.

If you read in an X10-format bitmap, the Quit and Write Output commands won't write out a new X11-format file unless you've changed at least one square on the bitmap. You can work around this by simply clearing a square and then clearing it back again.

This program would make a wonderful X Toolkit application.

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	Displays on the root window instead of creating a new window.
-d *pattern*	Specifies a bit pattern for drawing dashed lines for wire frames.
-i	Uses inverted colors for wire frames.
-dbl	Uses 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 8-bit-plane displays will probably be limited to eight colors when using double buffering.
-faces	Draws filled faces instead of wire frames.
-noedges	Doesn't draw wire frames. Typically used only when **-faces** is used.
-sleep *n*	Sleeps n seconds between each move of the object.
-obj *object*	Specifies what object to draw. If no object is specified, an icosahedron is drawn.
-objhelp	Prints out a list of the available objects, along with information about each object.
-colors *color color ...*	Specifies what colors should be used to draw the filled faces of the object. If fewer 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.

NAME

muncher – draws interesting patterns in an X window

SYNOPSIS

muncher [-option ...]

OPTIONS

–r	Displays in the root window.
–s *seed*	Seeds the random number seed.
–v	Runs in verbose mode.
–q	Runs in quite mode.
–geometry *geometry*	Defines the initial window geometry; see X(1).
–display *display*	Specifies 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 features lacking.

NAME

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

SYNOPSIS

plaid [-option ...]

OPTIONS

–b	Enables backing store for the window.
–geometry *geometry*	Defines the initial window geometry; see X(1).
–display *display*	Specifies 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 features lacking.

NAME

puzzle – 15-puzzle game for X

SYNOPSIS

puzzle [-option ...]

OPTIONS]

–display *display*	Specifies the display to use; see **X**(1).
–geometry *geometry*	Specifies the size and position of the puzzle window; see **X**(1).
–size *WIDTHxHEIGHT*	Specifies the size of the puzzle in squares.
–speed *num*	Specifies the speed in tiles per second for moving tiles around.
–picture *filename*	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	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.

NAME

twm - a window manager for X11 (Tom's Window Manager)

SYNTAX

twm [-display *display*]

DESCRIPTION

The *twm* program is a window manager client application of the window server.

The *twm* program was written to try and incorporate some of the desirable features of both the **wm** and **uwm** window managers. *Twm* puts a title bar on and re-parents each window. The title bar contains the window's name and three "buttons". When a pointer button press event is detected in any of these title bar "buttons" a certain action is performed. The left-most title bar button that looks like a window pane causes the window to be iconified. The right-most title bar button with the right-angles is the re-size button. The resize function is identical to the window resize function of the **wm** window manager. The other title bar button is supposed to represent a keyboard, a button click here causes the input focus to be directed to this window until the **f.unfocus** function is executed or another window is selected to get input focus. The title bar has the additional feature of becoming highlighted if the window has the input focus.

When *twm* is invoked, it attempts to read a *twm* startup file. The name of the *twm* startup file is:

$HOME/.twmrc

The *twm* startup file can be thought of as having three logical sections: the variables section, the buttons section, and the menus section. The variables section must come first, followed by either the buttons section or the menus section.

All variables and keywords may be entered in any combination of upper and lower case letters. Title functions and Root functions must be entered in lower case. A pound sign (#) character in the startup file indicates a comment which is terminated by the newline character. A *string* in the startup file is a series of characters enclosed by double quotes.

VARIABLES SECTION

Variables must be entered first, at the top of the startup file. Variables are initialized once when *twm* begins execution, they will not be effected when a **f.twmrc** function is executed.

Several variables take filenames as arguments. Filenames are processed as follows. *Twm* checks to see if the first character in the filename is a tilde (~), if it is, *twm* prepends the user's **HOME** environment variable to the filename. In the case of variables requiring bitmap files, if the above expansion does not produce a path to a valid bitmap file, the following steps are taken. If the **IconDirectory** variable has been set, and the filename does not start with a slash (/), the **IconDirectory** variable is prepended to the filename. If that path does not produce a valid bitmap file, the string "/usr/include/X11/bitmaps/" is prepended to the original filename.

The following describes the *twm* variables:

AutoRaise { *list* } This variable is a list of window names that will automatically raise to the top of the stacking order whenever the pointer enters the window. The window names in the list are the first characters in the window name to check for. For example:

```
AutoRaise
{
  "xterm"
  "xclock"
}
```

The above list contains two names which will match window names beginning with the string "xterm" or "xclock". The following window names will match and be in auto-raise mode: "xterm", "xterm_iguana", "xclock".

BorderColor *string* This variable sets the color of the border to placed around all non-iconified windows. It can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

BorderWidth *pixels* This variable specifies the width in pixels of of the border surrounding all windows. The default is 2.

Color { *colors* } This variable is a list of color assignments to be made if the default display has a depth greater than one, or in other words, has the ability to display more than black and white. For example:

```
Color
{
  BorderColor "red"
  TitleForeground "yellow"
  TitleBackground "blue"
}
```

The various color variables may be found in this section of the manual page. There is also a **Monochrome** list of colors that may be specified. This enables you to use the same initialization file on a color or monochrome display.

DontMoveOff If this variable is set, windows will not be allowed to be moved off the display.

ForceIcons This variable is only meaningful if a **Icons** list is defined. It forces the icon bitmaps listed in the **Icons** list to be used as window icons even if client programs supply their own icons. The default is to not force icons.

Icons { *list* } This variable is a list of window names and bitmap filenames to be used as icons. For example:

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

The names "xterm" and "xfd" are added to a list that is searched when the client window is reparented by *twm*. The window names specified are just the first portion of the name to match. In the above example, "xterm" would match "xtermfred" and also "xterm blob". The client window names are checked against those specified in this list in addition to the class name of the client if it is specified. By using the class name, all xterm windows can be given the same icon by the method used above even though the names of the windows may be different.

IconBackground *string*
This variable sets the background color of icons. It can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

IconBorderColor *string*
This variable sets the color of the border around icons. It can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

IconDirectory *string* This variable names the directory in which to search for icon bitmap files. This variable is described under the **VARIABLES SECTION** heading. The default is to have no icon directory.

IconFont *string* This variable names the font to be displayed within icons. The default is "8x13".

IconForeground *string* This variable sets the foreground color of icons. It can only be specified inside of

a **Color** or **Monochrome** list. The default is "black".

MenuBackground *string*
This variable sets the background color of menus. It can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

MenuFont *string* This variable names the font to be displayed within menus. The default is "8x13".

MenuForeground *string*
This variable sets the foreground color of menus. It can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

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

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

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

Monochrome { *colors* }
This variable is a list of color assignments to be made if the default display has a depth equal to one, or in other words can only display black and white pixels. For example:

```
Monochrome
{
  BorderColor "black"
  TitleForeground "black"
  TitleBackground "white"
}
```

The various color variables may be found in this section of the manual page. There is also a **Color** list of colors that may be specified. This enables you to use the same initialization file on a color or monochrome display.

NoTitle { *list* } This variable is a list of window names that will NOT have a title bar created for them. If the client does not get ButtonPress events, *twm* will get them and all Title functions currently in effect will work when pointer buttons are pressed in the client window. The list of windows and how they match window names is exactly like the **AutoRaise** variable described above.

NoRaiseOnDeiconify If this variable is specified, windows will not be raised to the top of the stacking order when de-iconified.

NoRaiseOnMove If this variable is specified, windows will not be raised to the top of the stacking order following a move.

NoRaiseOnResize If this variable is specified, windows will not be raised to the top of the stacking order following a resize.

NoTitleFocus If this variable is specified, input focus will not be directed to windows when the pointer is in the title bar. The default is to focus input to a client when the pointer is in the title bar.

ResizeFont *string* This variable names the font to be displayed in the dimensions window during window resize operations. The default is "fg-22".

TitleFont *string* This variable names the font to be displayed within the window title bar. Note that the title bar is only 17 pixels in height, so the largest practical font would be something like "9x15". The default is "8x13".

ReverseVideo This variable causes *twm* to display white characters on a black background, rather than black characters on white. This variable doesn't really do much now that you can specify individual colors.

TitleBackground *string*
This variable sets the background color for the title bars. It can only be specified inside of a **Color** or **Monochrome** list. The default is "white".

TitleForeground *string* This variable sets the foreground color for the title bars. It can only be specified inside of a **Color** or **Monochrome** list. The default is "black".

UnknownIcon *string* This variable specifies the file name of a bitmap format file to be used as the default icon. This bitmap will be used for the icon of all clients which do not provide an icon bitmap and are not listed int the **Icons** list. The default is to use no bitmap.

WarpCursor This variable causes the pointer cursor to be warped to a window which is being deiconified. The default is to not warp the cursor.

Zoom This variable causes a series of outlines to be drawn when a window is iconified or deiconified. The default is to not draw the outlines.

BUTTONS SECTION

This section deals with assignment of window manager functions to mouse buttons. The discussion of Root functions and Title functions is included here for historical reasons and to maintain compatibility with versions of *twm* prior to Revision 2.15. The section titled **New Button Specifications** describes the new syntax for defining pointer button sequences. The older syntaxes described in the following paragraphs are still supported.

The buttons section of the startup file contains definitions of functions to perform when pointer buttons are pressed. There are two classes of functions that can be tied to a pointer button: Title functions, which will be executed if a pointer button is pressed while the pointer is in the title bar of a window; and Root functions, which will be executed while the pointer is in the root window or if the client program is not processing pointer button events.

Title functions are assigned as follows:

TitleButton*n t.function* **TitleButton** is a startup file keyword. The *n* following **TitleButton** can be a number between 1 and 5 to indicate which pointer button the function is to be tied to. *t.function* may be any one of the following title functions.

Title Functions

The following title functions are still included for compatibility reasons. Since Revision 2.15, Itwm can execute any root function tied to a title bar pointer button press.

t.lower This function lowers the window to the bottom of the stacking order.

t.move This function causes a grid to appear over the window which can be moved to where you want the window to be moved. Double clicking the pointer button tied to this function causes a constrained move function to be executed. The pointer will be warped to the center of the grid. Moving the pointer to one of the grid lines will cause the window to begin moving in either an up-down motion or a left-right motion depending on which grid line the pointer was moved across.

t.nop This function does nothing.

t.raise This function raises the window to the top of the stacking order.

The defaults for title functions are as follows:

TitleButton1 t.raise

TitleButton2 t.move

TitleButton3 t.lower

TitleButton4 t.nop

TitleButton5 t.nop

Root functions are assigned either to a pointer button or a menu entry. The **f.title** function is the only one that doesn't really make sense to use as just a button function. It was designed to be used in a pull-down menu. Root functions are assigned to pointer buttons as follows:

Button*n function* — **Button** is a startup file keyword. The *n* following **Button** can be a number between 1 and 5 to indicate which pointer button the function is to be tied to. *function* may be any one of the following root functions.

New Button Specifications

Versions of *twm* newer or equal to Revision 2.15 support the new pointer button specification syntax which allows functions and menus to be tied to buttons with a variety of modifier keys being pressed in conjunction with pointer buttons. The syntax of the new button specification is:

Button*n* = *keys* : *context* : *function*

The *n* following **Button** can be a number between 1 and 5 to indicate which pointer button the function is to be tied to. The **keys** field is used to specify which modifier keys must be pressed in conjunction with the pointer button. The **keys** field may contain any combination of the letters **s**, **c**, and **m**, which stand for Shift, Control, and Meta, respectively. The **context** field specifies the context in which to look for the button press. Valid contexts are: **icon**, **root**, **title**, and **window**. The **function** field specifies what window manager function to perform. The old **Button** and **TitleButton** specifications are simple cases of the new syntax. Now for some examples:

```
Button2 =    : title : f.move            # 1
Button1 =    : root  : f.menu "menu 1"    # 2
Button1 = m  : icon  : f.menu "icon menu 1"  # 3
Button3 = msc : window : f.menu "menu3 1"    # 4
```

Line 1 specifies that when pointer button 2 is pressed in the title bar with no modifier keys pressed, the **f.move** function is to be executed. This is exactly the same as **TitleButton2 t.move**. Line 2 specifies that when pointer button 1 is pressed in the root window with no modifier keys pressed, the menu "menu 1" is popped up. This is exactly the same as **Button2 f.menu "menu 1"**. Line 3 specifies that when pointer button 1 is pressed in an icon window with the meta key pressed, the menu "icon menu 1" is popped up. Line 4 specifies that when pointer button 3 is pressed in a client window with the shift, control, and meta keys pressed, the menu "menu 3" is popped up.

Root Functions

! *string* — This function causes *string* to be sent to /bin/sh for execution.

^ *string* — This function causes *string* followed by a new line character to be placed in the window server's cut buffer.

f.circledown — This function causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup — This function raises the lowest window that is obscured by other windows.

f.cutfile — This function takes the contents of the window server's cut buffer and uses it as a filename to read into the server's cut buffer.

f.destroy — This function allows you to destroy a window client. The cursor is changed to a

skull and crossbones and the next window to receive a button press will be destroyed.

f.file *string* — This function assumes *string* is a file name. This file is read into the window server's cut buffer.

f.focus — This function implements the same function as the keyboard focus button in the title bar. The cursor is changed to a dot and the next window to receive a button press will gain the input focus.

f.iconify — This function implements the same function as the iconify button in the title bar. The cursor is changed to a dot and the next window to receive a button press will be iconified or de-iconified depending on the current state of the window.

f.lower — This function implements the window lower function of **t.lower**, but lets you get to it from a menu selection. The cursor is changed to a dot and the next window that receives a button press will be the window that is lowered.

f.menu *string* — This function assigns the pull-down menu named *string* to a pointer button. If this function is used as an entry in a pull-down menu a pull-right menu will be assigned to the menu entry.

f.move — This function implements the window move function of **t.move**, but lets you get to it from a menu selection. The cursor is changed to a double arrow and the next window that receives a button press will be the window that is moved.

f.nop — This function does nothing.

f.quit — This function causes *twm* to exit.

f.raise — This function implements the window raise function of **t.raise**, but lets you get to it from a menu selection. The cursor is changed to a dot and the next window that receives a button press will be the window that is raised.

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

f.resize — This function implements the window resize function of the resize button in the title bar. The cursor is changed to a double arrow and the next window that receives a button press will be the window that is resized.

f.source *string* — This function assumes *string* is a file name. The file is read and parsed as a *twm* startup file. This function is intended to be used only to re-build pull-down menus. None of the *twm* variables are changed.

f.title — This function is to be used as an entry in a pull-down menu. It centers the menu entry string in a menu entry and outlines it with a border. This function may be used more than once in a pull-down menu.

f.twmrc — This function causes the *$HOME/.twmrc* file to be re-read. This function is exactly like the **f.source** function without having to specify the filename.

f.unfocus — This function assigns input focus to the root window.

f.version — This function causes the *twm* version window to be displayed. This window will be displayed until a pointer button is pressed or the pointer is moved from one window to another.

f.winrefresh — This function is similar to the **f.refresh** function, but allows you to refresh a single window. The cursor is changed to a dot and the next window that receives a button press will be the window that is refreshed.

MENUS SECTION

The menus section is where pull-down menus are defined. Entries in menus consist of Root functions as described in the Buttons Section. The syntax to define a menu is:

Menu "*menu name*"
{
string *function*
string *function*
.
.
string *function*
}

The *menu name* should be an identical string to one being used with an **f.menu** Root function. Note that the *menu name* is case sensative. The *string* portion of each menu entry will be the text which will appear in the menu. The *function* portion of the menu entry is one of the Root functions described in the previous section.

WINDOW STARTUP

When a client is started, *twm* allows you to position and change the size of the window if the client has not specified an initial geometry. If the client has not specified both **User Specified Size** hints and **User Specified Position** hints, *twm* will put up a rubberband box indicating the initial window size. If pointer button one is pressed, the client window is created with the window position equal to the current pointer position. If pointer button two is pressed, *twm* allows the window to be resized. The resizing operation takes place until button two is released. While the initial positioning of the window is taking place, *twm* will place a window in the upper-left corner of the display showing the window's name. If resizing is taking place, *twm* will also place a window in the upper-left corner, indicating the current window size.

EXAMPLES

The following is an example *twm* startup file:

```
#************************************************************************
#
#   .twmrc
#
#************************************************************************

WarpCursor
BorderWidth     2
TitleFont       "8x13"
MenuFont        "8x13"
IconFont        "8x13"

Color
{
  BorderColor "red"
  TitleForeground "white"
  TitleBackground "blue"
  MenuForeground "yellow"
  MenuBackground "darkgreen"
  MenuTitleForeground "red"
  MenuTitleBackground "blue"
  IconForeground "darkgreen"
  IconBackground "cadetblue"
```

```
    IconBorderColor "green"
}

#Button = KEYS : CONTEXT : FUNCTION
#-----------------------------------
Button1 =     : root   : f.menu "button1"
Button2 =     : root   : f.menu "button2"
Button3 =     : root   : f.menu "button3"
Button1 =  m  : window : f.menu "button1"
Button2 =  m  : window : f.menu "button2"
Button3 =  m  : window : f.menu "button3"
Button1 =  m  : title  : f.menu "button1"
Button2 =  m  : title  : f.menu "button2"
Button3 =  m  : title  : f.menu "button3"
Button1 =     : title  : f.raise
Button2 =     : title  : f.move
Button3 =     : title  : t.lower

ForceIcons
IconDirectory     "~/icons"
Icons
{
  "xterm"         "xterm.icon"   # obtained from IconDirectory
  "xfd"           "xfd_icon"       # obtained from /usr/include/X11/bitmaps
}
UnknownIcon       "default.icon"

AutoRaise
{
  "xterm"         # all of my xterm windows will auto-raise
}

NoTitle
{
  "xclock"        # don't need a title bar on this ...
  "xckmail"       # or this
}.

menu "button1"
{
"Sun Systems"     f.title
"iguana"          !"xterm -T iguana =80x24+100+100 -e rlogin iguana &"
"worm"            !"xterm -T worm =80x24+100+100 &"
"shiva"           !"xterm -T shiva =80x24+200+200 -e rlogin shiva &"
"tegus"           !"xterm -T tegus =80x24+200+200 -e rlogin tegus &"
"Vax Systems"     f.title
"shade"           !"xterm -T shade =80x24+200+200 -e rlogin shade &"
"bilbo"           !"xterm -T bilbo =80x24+250+250 -e rlogin bilbo &"
"frodo"           !"xterm -T frodo =80x24+300+300 -e rlogin frodo &"
"esunix"          !"xterm -T esunix =80x24+350+350 -e rlogin esunix &"
"lynx8"           !"xterm -T lynx8 =80x24+390+390 -e rlogin lynx8 &"
}
```

```
menu "button2"
{
"Window Ops"            f.title
"Refresh"               f.refresh
"Focus on Root"         f.unfocus
"Source .twmrc"         f.twmrc
"Source something"      f.source "something"
"twm Version"           f.version
"(De)Iconify"       f.iconify
"Move Window"        f.move
"Resize Window"      f.resize
"Raise Window"       f.raise
"Lower Window"       f.lower
"Focus on Window"    f.focus
"Destroy Window"     f.destroy
}

menu "button3"
{
"Cut Buffer"            f.title
"Procedure Header"      f.file "/usr/ias_soft/tlastrange/src/proc.twm"
"File Header"           f.file "/usr/ias_soft/tlastrange/src/file.twm"
"pull right"            f.menu "blob"
}

menu "blob"
{
"pull right"            f.menu "final"
"another"               ^"some text"
}.

menu "final"
{
"entry 1"               f.nop
"entry 2"               f.nop
"entry 3"               f.nop
"entry 4"               f.nop
}
```

BUGS

Pull-right menus may still have some problems. They may sometimes stay around when all pointer buttons have been released.

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

The window auto-raise feature does not work consistently when the mouse is moved very fast over auto-raise windows.

There is a latency problem. You can move the mouse and begin typing and the first few characters will appear in the window the cursor was moved out of.

FILES

$HOME/.twmrc

SEE ALSO

X(1), Xserver(1)

COPYRIGHT

AUTHOR

Thomas E. LaStrange, Evans & Sutherland, Interactive Systems Division, Salt Lake City, Utah.

{ihnp4,decvax}!decwrl!esunix!tlastran
{cbosgd,ulysses}!utah-cs!esunix!tlastran

NAME

uwm – a window manager for X

SYNTAX

uwm [-display *display*] [-f *filename*]

DESCRIPTION

The **uwm** program is a window manager client application of the window server.

When **uwm** is invoked, it searches a predefined search path to locate any **uwm** startup files. If no startup files exist, **uwm** initializes its built-in default file.

If startup files exist in any of the following locations, it adds the variables to the default variables. In the case of contention, the variables in the last file found override previous specifications. Files in the **uwm** search path are:

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

To use only the settings defined in a single startup file, include the following variables at the top of that specific startup file: **resetbindings**, **resetmenus**, and **resetvariables**.

OPTIONS

–display *display*	Specifies the server to be used. See X(1) for details.
–f *filename*	Names an alternate file as a **uwm** startup file.

STARTUP FILE VARIABLES

Variables are typically entered first, at the top of the startup file. By convention, **resetbindings**, **resetmenus**, and **resetvariables** head the list.

autoselect/noautoselect	Places the menu cursor in the first menu item. If unspecified, the menu cursor is placed in the menu header when the menu is displayed.
delta=*pixels*	Indicates the number of pixels the cursor is moved before the action is interpreted by the window manager as a command. (Also refer to the **delta** mouse action.)
freeze/nofreeze	Locks all other client applications out of the server during certain window manager tasks, such as move and resize.
grid/nogrid	Displays a finely-ruled grid to help you position an icon or window during resize or move operations.
hiconpad=*n*	Indicates the number of pixels needed to pad an icon horizontally. The default is five pixels.
hmenupad=*n*	Indicates the amount of space (in pixels), that each menu item is padded above and below in the text.
iconfont=*fontname*	Names the font that is displayed within icons. Font names for a given server can be obtained using **xlsfonts**(1).
maxcolors=*n*	Limits the number of colors the window manager can use in a given invocation. If set to zero, or not specified, **uwm** assumes no limit to the number of colors it can take from the colormap. **Maxcolors** counts colors as they are included in the file.
normali/nonormali	Places icons created with **f.newiconify** within the root window, even if they are placed partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.
normalw/nonormalw	Places a window created with **f.newiconify** within the root window, even if

it is placed partially off the screen. With **nonormalw** the window is placed exactly where the cursor leaves it.

push=*n* Moves a window *n* number of pixels, or a relative amount of space, depending on whether **pushabsolute** or **pushrelative** is specified. Use this variable in conjunction with **f.pushup, f.pushdown, f.pushright**, or **f.pushleft.**

pushabsolute/pushrelative **Pushabsolute** indicates that the number entered with push is equivalent to pixels. When an **f.push** (left, right, up, or down) function is called, the window is moved exactly that number of pixels.

Pushrelative indicates that the number entered with the push variable represents a relative number. When an **f.push** function is called, the window is invisibly divided into the number of parts you entered with the push variable, and the window is moved one part.

resetbindings, resetmenus, resetvariables
Resets all previous function bindings, menus, and variables entries, specified in any startup file in the **uwm** search path, including those in the default environment. By convention, these variables are entered first in the startup file.

resizefont=*fontname* Identifies the font of the indicator that displays in the corner of the window as you resize windows. See **xlsfonts**(1) for obtaining font names.

reverse/noreverse Defines the display as black characters on a white background for the window manager windows and icons.

viconpad=*n* Indicates the number of pixels to pad an icon vertically. Default is five pixels.

vmenupad=*n* Indicates the amount of space in pixels that the menu is padded on the right and left of the text.

volume=*n* Increases or decreases the base level volume set by the **xset**(1) command. Enter an integer from 0 to 7, 7 being the loudest.

zap/nozap Causes ghost lines to follow the window or icon from its previous default location to its new location during a move or resize operation.

BINDING SYNTAX

"function=[control key(s)]:[context]:mouse events:" menu name ""

The function and mouse events are required input. The menu name is required with the **f.menu** function definition only.

Function

f.beep Emits a beep from the keyboard. Loudness is determined by the volume variable.

f.circledown Causes the top window that is obscuring another window to drop to the bottom of the stack of windows.

f.circleup Exposes the lowest window that is obscured by other windows.

f.continue Releases the window server display action after you stop action with the **f.pause** function.

f.focus Directs all keyboard input to the selected window. To reset the focus to all windows, invoke **f.focus** from the root window.

f.iconify When implemented from a window, this function converts the window to its respective icon. When implemented from an icon, **f.iconify** converts the icon to its respective window.

f.lower Lowers a window that is obstructing a window below it.

f.menu Invokes a menu. Enclose "menu name" in quotes if it contains blank characters or parentheses. **f.menu=**[*control key(s)*]:[*context*]:*mouse events*:" *menu name* "

f.move Moves a window or icon to a new location, which becomes the default location.

f.moveopaque Moves a window or icon to a new screen location. When using this function, the entire window or icon is moved to the new screen location. The grid effect is not used with this function.

f.newiconify Allows you to create a window or icon and then position the window or icon in a new default location on the screen.

f.pause Temporarily stops all display action. To release the screen and immediately update all windows, use the **f.continue** function.

f.pushdown Moves a window down. The distance of the push is determined by the push variables.

f.pushleft Moves a window to the left. The distance of the push is determined by the push variables.

f.pushright Moves a window to the right. The distance of the push is determined by the push variables.

f.pushup Moves a window up. The distance of the push is determined by the push variables.

f.raise Raises a window that is being obstructed by a window above it.

f.refresh Results in exposure events being sent to the window server clients for all unobscured or partially obscured windows. The windows do not refresh correctly if the exposure events are not handled properly.

f.resize Resizes an existing window. Note that some clients, notably editors, react unpredictably if you resize the window while the client is running.

f.restart Causes the window manager application to restart, retracing the **uwm** search path and initializing the variables it finds.

Control Keys

By default, the window manager uses meta as its control key. It can also use ctrl, shift, lock, or null (no control key). Control keys must be entered in lowercase, and can be abbreviated as: c, l, m, s for ctrl, lock, meta, and shift, respectively.

You can bind one, two, or no control keys to a function. Use the bar (|) character to combine control keys.

Note that client applications other than the window manager use the shift as a control key. If you bind the shift key to a window manager function, you can not use other client applications that require this key.

Context

The context refers to the screen location of the cursor when a command is initiated. When you include a context entry in a binding, the cursor must be in that context or the function will not be activated. The window manager recognizes the following four contexts: icon, window, root, (null).

The root context refers to the root, or background window, A (null) context is indicated when the context field is left blank, and allows a function to be invoked from any screen location. Combine contexts using the bar (|) character.

Mouse Buttons

Any of the following mouse buttons are accepted in lowercase and can be abbreviated as l, m, or r, respectively: left, middle, right.

With the specific button, you must identify the action of that button. Mouse actions can be:

down Occurs when the specified button is pressed down.

up Occurs when the specified button is released.

delta Indicates that the mouse must be moved the number of pixels specified with the delta variable before the specified function is invoked. The mouse can be moved in any direction to satisfy the delta requirement.

MENU DEFINITION

After binding a set of function keys and a menu name to **f.menu**, you must define the menu to be invoked, using the following syntax:

menu = " *menu name* " {
"*item name*" : "*action*"
.
.
.
}

Enter the menu name exactly the way it is entered with the **f.menu** function or the window manager will not recognize the link. If the menu name contains blank strings, tabs or parentheses, it must be quoted here and in the **f.menu** function entry. You can enter as many menu items as your screen is long. You cannot scroll within menus.

Any menu entry that contains quotes, special characters, parentheses, tabs, or strings of blanks must be enclosed in double quotes. Follow the item name by a colon (:).

Menu Action

Window manager functions
: Any function previously described (e.g., **f.move** or **f.iconify**).

Shell commands
: Begins with an exclamation point (!) and is set to run in background. You cannot include a new line character within a shell command.

Text strings
: Text strings are placed in the window server's cut buffer.

Strings starting with an up arrow (^) have a new line character appended to the string after the up arrow (^) is stripped from it.

Strings starting with a bar character (|) are copied as is after the bar character (|) is stripped.

Color Menus

Use the following syntax to add color to menus:

menu = "*menu name*" (*color1*:*color2*:*color3*:*color4*) {
"*item name*" : (*color5* :*color6*) : " *action* "
.
.
.
}

color1 Foreground color of the header.

color2 Background color of the header.

color3 Foreground color of the highlighter, the horizontal band of color that moves with the cursor within the menu.

color4 Background color of the highlighter.

color5 Foreground color for the individual menu item.

color6 Background color for the individual menu item.

Color Defaults

Colors default to the colors of the root window under any of the following conditions:

1) If you run out of colormap entries, either before or during an invocation of **uwm**.

2) If you specify a foreground or background color that does not exist in the RGB color database of the server (see */usr/lib/X11/rgb.txt* for a sample), both the foreground and background colors default to the root window colors.

3) If you omit a foreground or background color, both the foreground and background colors default to the root window colors.

4) If the total number of colors specified in the startup file exceeds the number specified in the *maxcolors* variable.

5) If you specify no colors in the startup file.

EXAMPLES

The following sample startup file shows the default window manager options:

```
# Global variables
#
resetbindings;resetvariables;resetmenus
autoselect
delta=25
freeze
grid
hiconpad=5
hmenupad=6
iconfont=oldeng
menufont=timrom12b
resizefont=9x15
viconpad=5
vmenupad=3
volume=7
#
# Mouse button/key maps
#
# FUNCTION   KEYS  CONTEXT  BUTTON   MENU(if any)
# ========  ====  =======  ======  ============
f.menu =     meta :    :left down  :"WINDOW OPS"
f.menu =     meta :    :middle down :"EXTENDED WINDOW OPS"
f.move =     meta :wli  :right down
f.circleup = meta :root :right down
#
# Menu specifications
#
menu = "WINDOW OPS" {
"(De)Iconify":   f.iconify
Move:            f.move
Resize:          f.resize
Lower:           f.lower
```

```
Raise:              f.raise
}

menu = "EXTENDED WINDOW OPS" {
Create Window:                  !"xterm &"
Iconify at New Position:    f.lowericonify
Focus Keyboard on Window:       f.focus
Freeze All Windows:             f.pause
Unfreeze All Windows:           f.continue
Circulate Windows Up:           f.circleup
Circulate Windows Down:         f.circledown
}
```

RESTRICTIONS

The color specifications have no effect on a monochrome system.

FILES

/usr/lib/X11/uwm/system.uwmrc
$HOME/.uwmrc

SEE ALSO

X(1), **Xserver**(1), **xset**(1), **xlsfonts**(1)

NAME

wm – a simple real-estate-driven window manager

SYNOPSIS

wm [-display *display*]

DESCRIPTION

Wm is a very primitive overlapping window manager for X11. It was developed to help with the debugging of the X11 server; we do not suggest that the user interface presented here is a desired one, and we do not suggest that you try to use this program on a regular basis.

Wm decorates each mapped application window with a banner. The banner consists of four fields. Left-to-right, they are:

Circulate button - A command button which causes the window to change its position in the stacking order.

Title region - An area in which an applications name or other specified information is displayed. It is also used by the user to move the window.

Iconize button - A command button which causes the window to be replaced by an icon.

Resize button - A command button which allows the user to change the size of the window.

Wm supports the following user actions:

Raising or lowering a window in the stack of windows.

Locating the pointer cursor in the Circulate box of a partially obscured window and clicking with any pointer button raises this window to the top of the stack of windows so that it is no longer obscured. Locating a pointer cursor in the Circulate box of a window which is currently on top of the window stack sends the window to the bottom of the stack.

Iconizing a window.

Locating the pointer cursor in the Iconize box and clicking any pointer button causes the window to be unmapped and the associated icon to become mapped. The icon appears at its last location, or, if this window has never been iconized, under the cursor. However, if the client program initially set an icon position in the WM_HINTS property, then that icon position is used instead as the initial icon position. To position an icon while iconizing the window, locate the cursor in the Iconize box and press down any pointer button. A rubber-band outline of the icon appears under the cursor. While holding down the pointer button, drag the cursor to the desired location for the icon. The outline follows the cursor on the screen. When the outline moves to the desired location for the icon, release the pointer button. The client window becomes unmapped, and its icon appears at the desired location. To cancel this operation while the pointer button is down, click another pointer button.

Deiconizing an icon.

Locating the pointer cursor in an icon and clicking any pointer button causes the icon to be unmapped, and the associated window to become mapped. To cancel this operation while the pointer button is down, click another pointer button.

Moving a window on the screen.

Locating the pointer cursor in the area of the title region and pressing any pointer button causes a "rubber-band" outline of the window to appear. As you move (drag) the cursor (while holding down the pointer button), the outline moves accordingly. When the button is released, the window is repainted in the last location of the rubber-band outline. If you press another pointer button during the drag, the operation is cancelled, the rubber-band outline disappears, and the window is not moved. Note that a portion of the title region is constrained to remain on the screen.

Resizing a window.

Locating the pointer cursor in the resize box and pressing any pointer button initiates the spring-loaded resize mode. Then as soon a the cursor touches a border (while the pointer button is down), that border becomes a rubber-band line which follows the cursor until the button is released. If the

cursor then touches an adjacent border, that border also becomes a rubber-band line, and the window can be resized in two dimensions at once. If the cursor touches a border after having touched the opposite border, the first border touched reverts to its original location, and the other one becomes a rubber-band line which follows the cursor. If you press another pointer button during the drag, the operation is cancelled, the rubber-band outline disappears, and the window does not change size. Note that the pointer cursor has to touch a border to initiate the resize action. As in the move operation, a protion of the title region is constrained to remain on the screen.

Moving an icon on the screen

To move an icon, press the Shift key and hold it, then position the pointer cursor in the icon, press any pointer button, and proceed dragging an outline of the icon around by moving the pointer cursor (with the pointer button down). When the outline moves to the desired position, release the pointer button and the Shift key. To cancel, click another pointer button during the drag; the icon will not move.

NOTES FOR CLIENT PROGRAMS

Wm uses the WM_ICON_NAME, WM_NAME, and WM_HINTS properties. It keeps the name in the Title region updated as the WM_NAME property changes. It keeps the name in the icon updated as the WM_ICON_NAME property changes; if a client does not set the WM_ICON_NAME property, **wm** will use the WM_NAME property for the icon name. **Wm** allows only text icons, and sets the icon sizes to accommodate the icon name. The maximum name length for both the icon name and the Title region name is 100 characters.

Of the WMHints, **wm** ignores all but icon_x and icon_y, which it uses for initial icon placement. These need to be set by the client before its window is mapped, because **wm** reads them only once, when it first encounters the window.

SEE ALSO

X(1), *Inter-Client Communication Conventions Manual*

BUGS

This program does not necessarily implement the current window manager protcols.

DIAGNOSTICS

If you try to run **wm** while you are already running a window manager, **wm** will let you know.

NAME

x10tox11 – X version 10 to version 11 protocol converter

SYNTAX

x10tox11 [-display *host:display*]

DESCRIPTION

x10tox11 masquerades as an X Window System Version 10 server. It enables an X Version 10 client to run unchanged under X Version 11 by converting Version 10 requests into appropriate Version 11 requests, and by converting all Version 11 events received from the server into Version 10 events. From the perspective of Version 10 clients, all Version 11 clients look like Version 10 clients; and from the perspective of Version 11 clients, all Version 10 clients look like Version 11 clients. Hence, a Version 11 window manager can manipulate Version 10 clients.

This program does NOT use the X10 *libnest* ddx library. It does actual protocol translation, rather than simply using X11 graphics calls to implement X10 low level operations. As a result, it is both faster and more robust than the X10 Xnest server.

OPTIONS

–display *host:display* Specifies the host and server to use.

SEE ALSO

X(1), **Xserver**(1)

BUGS

There are limitations with respect to emulating Version 10 through a Version 11 server. See the README file in the sources for more details.

NAME

xbiff – mailbox flag for X

SYNTAX

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. 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 Indicates that a brief summary of the allowed options should be printed on the standard error.

–update *seconds* 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.

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

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

–bg *color* Specifies the color to use for the background of the window. The default is white.

–bd *color* Specifies the color to use for the border of the window. The default is black.

–bw *number* Specifies the width in pixels of the border surrounding the window.

–fg *color* Specifies the color to use for the foreground of the window. The default is black.

–rv Indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry* Specifies the prefered size and position of the mailbox window; see X(1).

–display *display* Specifies the X server to contact; see X(1).

–xrm *resourcestring* 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:

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.

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.

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.

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.

NAME

xcalc – scientific calculator for X

SYNTAX

xcalc [-display *display*] [-bw *pixels*] [-stip] [-rv] [-rpn] [-analog] [-geometry *geometry*]

DESCRIPTION

Xcalc is a scientific calculator desktop accessory that can emulate a TI-30, an HP-10C, and a slide rule.

OPTIONS

–bw *pixels*	Specifies the border width in pixels.
–stip	Indicates that the background of the calculator should be drawn using a stipple of the foreground and background colors. On monochrome displays this makes for a nicer display.
–rv	Indicates that reverse video should be used.
–rpn	Indicates that Reverse Polish Notation should be used. In this mode the calculator will look and behave like an HP-10C. Without this flag, it will emulate a TI-30.
–analog	Indicates that a slide rule should be used.
–geometry *geometry*	Specifies the size and placement of the top level window. By default, the minimum size will be used. Note that your window manager may require you to place it explicitly anyway.
–display *display*	Specifies the X server to contact; see X(1).

OPERATION

Pointer Operation: The left mouse button is the only one used to operate the calculator. Pressing the AC key with the right mouse button terminates the calculator.

Key Usage (Normal mode):

The number keys, the +/- key, and the +, -, *, /, and = keys all do exactly what you would expect them to. It should be noted that the operators obey the standard rules of precedence. Thus, entering "3+4*5=" results in "23," not "35." The parentheses can be used to override this. For example, "(1+2+3)*(4+5+6)=" results in "6*15=90." The non-obvious keys are detailed below.

1/x	Replaces the number in the display with its reciprocal.
x^2	Squares the number in the display.
SQRT	Takes the square root of the number in the display.
CE/C	When pressed once, clears the number in the display without clearing the state of the machine. Allows you to re-enter a number if you screw it up. Pressing it twice clears the state, also.
AC	Clears everything: the display, the state, and the memory. Pressing it with the right mouse button turns off the calculator, in that it exits the program.
INV	Inverts the meaning of the function keys. See the individual function keys for details.
sin	Computes the sine of the number in the display, as interpreted by the current DRG mode (see DRG below). If inverted, it computes the arcsine.
cos	Computes the cosine, or arccosine when inverted.
tan	Computes the tangent, or arctangent when inverted.

DRG Changes the DRG mode, as indicated by "DEG," "RAD," or "GRAD" at the bottom of the 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 gradians. When inverted, the DRG key converts degrees to radians to gradians and vice-versa. Example: put the calculator into "DEG" mode, and type "45 INV DRG." The display should now show something along the lines of ".785398," which is 45 degrees converted to radians.

e The constant "e" (2.7182818...).

EE Used for entering exponential numbers. For example, to enter "-2.3E-4" you'd type "2 . 3 +/- EE 4 +/-."

log Calculates the log (base 10) of the number in the display. When inverted, it raises "10.0" to the number in the display. For example, typing "3 INV log" should result in "1000."

ln Calcuates the log (base e) of the number in the display. When inverted, it raises "e" to the number in the display. For example, typing "e ln" should result in "1."

y^x Raises the number on the left to the power of the number on the right. For example "2 y^x 3 =" results in "8," which is 2^3. For a further example, "(1+2+3) y^x (1+2) =" equals "6 y^x 3" which equals "216."

PI The constant "pi" (3.1415927...).

x! Computes the factorial of the number in the display. The number in the display must be an integer in the range 0-500, though, depending on your math library, it might overflow long before that.

STO Copies the number in the display to the memory location.

RCL Copies the number from the memory location to the display.

SUM Adds the number in the display to the number in the memory location.

EXC Swaps the number in the display with the number in the memory location.

Key Usage (RPN mode):

The number keys, CHS (change sign), +, -, *, /, and ENTR keys all do exactly what you would expect them to. Many of the remaining keys are the same as in normal mode. The differences are detailed below.

<- Is a backspace key that can be used while typing a number. It will erase digits from the display.

ON Clears everything: the display, the state, and the memory. Pressing it with the right button "turns off" the calculator, in that it exits the program.

INV Inverts the meaning of the function keys. This would be the "f" key on an HP calculator, but **xcalc** does not have the resolution to display multiple legends on each key. See the individual function keys for details.

10^x Raises "10.0" to the number in the top of the stack. When inverted, it calculates the log (base 10) of the number in the display.

e^x Raises "e" to the number in the top of the stack. When inverted, it calcuates the log (base e) of the number in the display.

STO Copies the number in the top of the stack to a memory location. There are 10 memory locations. The desired memory is specified by following the pressing of this key with the pressing of a digit key.

RCL	Pushes the number from the specified memory location onto the stack.
SUM	Adds the number on top of the stack to the number in the specified memory location.
x:y	Exchanges the numbers in the top two stack positions.
R v	Rolls the stack downward. When inverted, it rolls the stack upward.
blank	These keys were used for programming functions on the HP11-C. Their functionality has not been duplicated here.

KEYBOARD EQUIVALENTS

If you have the pointer in the **xcalc** window, you can use the keyboard to speed entry, as almost all of the calculator keys have a keyboard equivalent. The number keys, the operator keys, and the parentheses all have the obvious equivalent. The less-obvious equivalents are as follows:

n: +/-	!: x!
p: PI	e: EE
l: ln	^: y^x
i: INV	s: sin
c: cos	t: tan
d: DRG	BS, DEL: CE/C ("<-" in RPN mode)
CR: ENTR	q: quit

COLOR USAGE

Xcalc uses a lot of colors, given the opportunity. In the default case, it just uses two colors (Foreground and Background) for everything. You can specify the colors used for the number keys, the operator (+-*/=) keys, the function keys, the display, and the icon.

X DEFAULTS

The program uses the routine **XGetDefault**(3X) to read defaults, so its resource names are all capitalized.

BorderWidth	Specifies the width of the border. The default is 2.
ReverseVideo	Indicates that reverse video should be used.
Stipple	Indicates that the background should be stippled. The default is on for monochrome displays, and off for color displays.
Mode	Specifies the default mode. Allowable values are rpn and analog.
Foreground	Specifies the default color used for borders and text.
Background	Specifies the default color used for the background.
NKeyFore, NKeyBack	Specifies the colors used for the number keys.
OKeyFore, OKeyBack	Specifies the colors used for the operator keys.
FKeyFore, FKeyBack	Specifies the colors used for the function keys.
DispFore, DispBack	Specifies the colors used for the display.
IconFore, IconBack	Specifies the colors used for the icon.

EXAMPLES

If you're running on a monochrome display, you shouldn't need any *.Xdefaults* entries for **xcalc**. On a color display, you might want to try the following in normal mode:

xcalc.Foreground:	Black
xcalc.Background:	LightSteelBlue
xcalc.NKeyFore:	Black
xcalc.NKeyBack:	White
xcalc.OKeyFore:	Aquamarine
xcalc.OKeyBack:	DarkSlateGray

xcalc.FKeyFore:	White
xcalc.FKeyBack:	#900
xcalc.DispFore:	Yellow
xcalc.DispBack:	#777
xcalc.IconFore:	Red
xcalc.IconBack:	White

SEE ALSO

X(1), **xrdb**(1), **XGetDefault**(3X)

BUGS

The calculator doesn't resize.

The slide rule may or may not work correctly.

This application should really be implemented with the X Toolkit. It would make a very good example of a compound widget.

NAME

xclock – analog / digital clock for X

SYNTAX

xclock [*-toolkitoption*] [*-option*]

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	Indicates that a brief summary of the allowed options should be printed on the standard error.
–analog	Indicates that a conventional 12-hour clock face with tick marks and hands should be used. This is the default.
–digital	Indicates that a 24-hour digital clock should be used.
–chime	Indicates that the clock should chime once on the half hour and twice on the hour.
–hd *color*	Specifies the color of the hands on an analog clock. The default is black.
–hl *color*	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*	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*	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.

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

–bg *color*	Specifies the color to use for the background of the window. The default is white.
–bd *color*	Specifies the color to use for the border of the window. The default is black.
–bw *number*	Specifies the width in pixels of the border surrounding the window.
–fg *color*	Specifies the color to use for displaying text. The default is black.
–fn *font*	Specifies the font to be used for displaying normal text. The default is 6x10.
–rv	Indicates that reverse video should be simulated by swapping the foreground and background colors.
–geometry *geometry*	Specifies the prefered size and position of the clock window.
–display *host:display*	Specifies the host and server to use.
–xrm *resourcestring*	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 *Clock* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)	Specifies the width of the clock.
height (class **Height**)	Specifies the height of the clock.
update (class **Interval**)	Specifies the frequency in seconds at which the time should be redisplayed.
foreground (class **Foreground**)	Specifies the color for the tick marks. Using the class specifies the color for all things that normally would appear in the foreground color. The default is black since the core default for background is white.
hand (class **Foreground**)	Specifies the color of the insides of the clock's hands.
high (class **Foreground**)	Specifies the color used to highlight the clock's hands.
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.
reverseVideo (class **ReverseVideo**)	Specifies that the foreground and background colors should be reversed.

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), **time**(3C), Athena Clock widget

BUGS

Xclock believes the system clock.

When in digital mode, the string should be centered automatically.

When specifying an offset, the grammar requires an hours field but if only minutes are given they will be quietly ignored. A negative offset of less than 1 hour is treated as a positive offset.

Digital clock windows default to the analog clock size.

Border color has to be explicitly specified when reverse video is used.

When the update is an even divisor of 60 seconds, the second hand should always be on a multiple of the update time.

NAME

xdpr – dump an X window directly to the printer

SYNTAX

xdpr [*-option*]

DESCRIPTION

Xdpr runs the commands **xwd**(1), **xpr**(1), and **lpr**(1) to dump an X window, process it for a laser printer, and print it out. This is the easiest way to get a printout of a window. **Xdpr**, by default, prints the largest possible representation of the window on the output page.

OPTIONS

–P*printer*	Specifies the name of the printer to be used.
–display *display*	Specifies the X server to contact; see **X**(1).

Any other arguments are passed as arguments to the **xpr**(1) command.

SEE ALSO

X(1), **xwd**(1), **xpr**(1), **xwud**(1), **lpr**(1)

ENVIRONMENT

DISPLAY	To get the default host and display number.

NAME

xedit – simple text editor for X

SYNTAX

xedit [*-toolkitoption*] [*filename*]

OPTIONS

Xedit accepts all of the standard X Toolkit command line options, plus:

filename — Specifies the file that is to be loaded during start-up. 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 three areas:

Commands Menu — Lists editing commands (for example, **Undo** or **Search**).

Message Window — Displays **xedit** messages. In addition, this window can be used as a scratch pad.

Edit Window — Displays the text of the file that you are editing or creating.

COMMANDS

Quit — Quits the current editing session. If any changes have not been saved, **xedit** displays a warning message and allows you to save the file.

Save — Stores a copy of the original, unedited file in *file*.BAK, then overwrites the original file with the edited contents.

Edit — Allows the text displayed in the Edit window to be edited.

Load — Loads the specified file and displays it in the Edit window.

Undo — Undoes the last edit only.

More — Undoes each edit previous to the last edit, which must first be undone with the **Undo** command.

Jump — Advances the cursor from the beginning of the file to the text line that corresponds to the selected line number.

<< — Searches from the cursor back to the beginning of the file for the string entered in the Search input box. If you do not enter a string in the Search input box, **xedit** automatically copies the last string that you selected from any X application into the Search input box and searches for that string.

Search >> — Searches from the cursor forward to the end of the file for the string entered in the search input box. If you do not enter a string in the Search input box, **xedit** automatically copies the last string that you selected from any X application into the Search input box and searches for that string.

Replace — Replaces the last searched-for string with the string specified in the Replace input box. If no string has been previously searched for, searches from the insert cursor to the end of the file for the next occurrence of the search string and highlights it. **All** Repositions the cursor at the beginning of the file and replaces all occurrences of the search string with the string specified in the Replace input box.

X DEFAULTS

For **xedit**, the available class identifiers are:

ButtonBox
Command
Scrollbar
Text

For **xedit**, the available name identifiers are:

All
Edit
EditWindow
Jump
Load
MessageWindow
More
Quit
Replace
Save
Undo
xedit

For **xedit**, the available resources are:

EnableBackups	Specifies that when edits made to an existing file are saved, **xedit** is to copy the original version of that file to *file.BAK* before it saves the changes. If the value of this option is specified as off, a backup file is not created.
background	Specifies the background color to be displayed in command buttons. The default is white.
border	Specifies the border color of the **xedit** window.
borderWidth	Specifies the border width, in pixels, of the **xedit** window.
font	Specifies the font displayed in the **xedit** window.
foreground	Specifies the foreground color of the **xedit** window. The default is black.
geometry	Specifies the geometry (window size and screen location) to be used as the default for the **xedit** window. For information about the format of the geometry specification, see ARGUMENTS.
internalHeight	Specifies the internal horizontal padding (spacing between text and button border) for command buttons.
internalWidth	Specifies the internal vertical padding (spacing between text and button border) for command buttons.

KEY BINDINGS

Each specification included in the *.XtActions* file modifies a key setting for the editor that **xedit** uses. When defining key specifications, you must use the following resource specification:

text.EventBindings: .XtActions

Each key specification assigns an editor command to a named key and/or mouse combination and has the format:

key: *function*

key	Specifies the key or mouse button that is used to invoke the named function.
function	Specifies the function to be invoked when the named key is pressed.

FILES

~/.XtActions
/usr/lib/X11/.XtActions

SEE ALSO

X(1), **xrdb**(1)

RESTRICTIONS

Large numbers of certain edit functions (for example, Undo or More) tend to degrade performance over time. If there is a noticeable decrease in response time, save and reload the file.

NAME

xfd – font displayer for X

SYNTAX

xfd [*-options*] [*fontname*]

OPTIONS

–bw *number*	Specifies the width of the window border in pixels.
–rv	Reverses the foreground and background. The default colors are black on white.
–fw	Overrides a previous choice of reverse video. The foreground and background colors will not be switched.
–fg *color*	determines the foreground color (the color of the text) on color displays.
–bg *color*	Determines the background color on color displays.
–bd *color*	Determines the color of the border on color displays.
–bf *fontname*	Specifies the font to be used for the messages at the bottom of the window.
–tl *title*	Specifies that the title of the displayed window should be *title*.
–in *iconname*	Specifies that the name of the icon should be *iconname*.
–icon *filename*	Specifies that the bitmap in file *filename* should be used for the icon.
–verbose	Specifies that verbose mode should be used.
–gray	Specifies that a gray background should be used.
–start *charnum*	Specifies that character number *charnum* should be the first character displayed.
–geometry *geometry*	Specifies an initial window geometry; see X(1).
–display *display*	Specifies the display to use; see X(1).

DESCRIPTION

Xfd creates a window in which the characters in the named font are displayed. The characters are shown in increasing order from left to right, top to bottom. The first character displayed at the top left is character number 0 unless the **-start** option is supplied in which case the character with the number given in the **-start** option is used.

The characters are displayed in a grid of boxes, each large enough to hold any character of the font. If the **-gray** option is supplied, the characters are displayed using **XDrawImageString**(3X) using the foreground and background colors on a gray background. This permits determining exactly how **XDrawImageString**(3X) draws any given character. If **-gray** is not supplied, the characters are simply drawn using the foreground color on the background color.

All the characters in the font may not fit in the window at once. To see additional characters, click the right mouse button on the window. This causes the next window full of characters to be displayed. Clicking the left mouse button on the window causes the previous window full of characters to be displayed. **Xfd** beeps if an attempt is made to go back past the 0th character.

Note that if the font is an 8-bit font, the characters 256-511 (0x100-0x1ff), 512-767 (0x200-0x2ff), ... displays exactly the same as the characters 0-255 (0x00-0xff). **Xfd,** by default, creates a window sufficiently large enough to display the first 256 characters using a 16x16 grid. In this case, there is no need to scroll window fulls forward or backward in order to see the entire contents of an 8-bit font. This window may not fit on the screen.

Clicking the middle button on a character causes that character's number to be displayed in both decimal and hexidecimal at the bottom of the window. If verbose mode is selected, additional information about that particular character is displayed as well. The displayed information includes the width of the character, its left bearing, right bearing, ascent, and descent. If verbose mode is selected, typing "<" or ">" into the

window displays the minimum or maximum values respectively taken on by each of these fields over the entire font.

The font name is interpreted by the X server. To obtain a list of all the fonts available, use **xlsfonts**(1).

If no font name is given on the command line, **Xfd** displays the font "fixed."

The window stays around until the **xfd** process is killed or "q," "Q," " ", or ctrl-c is typed into the **xfd** window.

X DEFAULTS

The **xfd** program uses the routine **XGetDefault**(3X) to read defaults, so its resource names are all capitalized.

BorderWidth	Sets the border width of the window.
BorderColor	Sets the border color of the window.
ReverseVideo	If "on," reverses the definition of foreground and background color.
Foreground	Sets the foreground color.
Background	Sets the background color.
BodyFont	Sets the font to be used in the body of the window (i.e., for messages, etc.). This is not the font that **xfd** displays, just the font it uses to display information about the font being displayed.
IconName	Sets the name of the icon.
IconBitmap	Sets the file we should look in to get the bitmap for the icon.
Title	Sets the title to be used.

SEE ALSO

X(1), **xlsfonts**(1), **xrdb**(1), **XDrawImageString**(3X), **XGetDefault**(3X)

ENVIRONMENT

DISPLAY	To get the default host and display to use.
XENVIRONMENT	To get the name of a resource file that overrides the global resources stored in the RESOURCE_MANAGER property.

BUGS

It should display the name of the font somewhere.

It should be rewritten to use the X Toolkit.

It should skip over pages full of non-existant characters.

NAME

xhost – server access control program for X

SYNTAX

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 are denied. Note that the current machine is allowed to be removed; however, further connections (including attempts to add it back) are not 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).

NAME

xinit – X Window System initializer

SYNTAX

xinit [[*client*] *options*] [-- [server] [display] *options*]

DESCRIPTION

The **xinit** program is used to start the X Window System server and a first client program (usually a terminal emulator) on systems that cannot start X directly from */etc/init* or in environments that use multiple window systems. When this first client exits, **xinit** kills the X server and then terminates.

Unless otherwise specified on the command line, **xinit** assumes that there are programs called "X" and "xterm" in the current search path. It starts the server on display 0 and then runs an **xterm** using the following command line:

```
xterm -geometry +1+1 -n login -display unix:0
```

(Systems that don't support UNIX domain sockets are started with *hostname:0* instead.)

An alternate client and/or server may be specified on the command line. The desired client program and its arguments should be given as the first command line arguments to **xinit**. To specify a particular server command line, append a double dash (--) to the **xinit** command line (after any client and arguments) followed by the desired server command.

Both the client program name and the server program name must begin with a slash (/) or a period (.). Otherwise, they are treated as an argument to be appended to their respective startup lines. This makes it possible to add arguments (for example, foreground and background colors) without having to retype the whole command line.

If an explicit server name is not given and the first argument following the double dash (--) is a digit, **xinit** uses that number as the display number instead of zero. All remaining arguments are appended to the server command line.

Since the default client is **xterm**(1), arguments entered after **xinit** are simply appended as further options to the default **xterm** command line.

EXAMPLES

```
xinit -geometry =80x65+10+10 -fn 8x13 -j -fg white -bg navy

xinit -e widgets -- Xsun -l -c

xinit rsh fasthost cpupig -display workstation:1 -- 1 -a 2 -t 5
```

SEE ALSO

X(1), **Xserver**(1), **xterm**(1)

NAME

xis – server for X.11

SYNTAX

xis [option] ...

DESCRIPTION

xis is the server for Version 11 of the X Window System on Integrated Solutions workstations. It is normally started by **xinit**(1), or by a shell script run from an interactive shell. Note that you probably need to give command line options to **xinit**(1) in order to get it to run theserver with the desired command line options, and to run **xterm**(1) with the desired options, display, etc. See the manpage for **xinit**(1).

CONFIGURATIONS

display workstation hardware. It requires a keyboard and mouse.

OPTIONS

The standard X11 server command line options are used. These options are described under **Xserver**(1).

You can exit the server by entering **exit** at the command line prompt in the console window. Note that this terminate the server but that there is no guarantee that all the existing processes will die. Refer to the *X11 Window Manager's Guide* for a more complete explanation.

FILES

/usr/bin/X11/Xis

SEE ALSO

Xserver(1)

NAME

xload – load average display for X

SYNTAX

xload [*-toolkitoption*] [-scale *integer*] [-update *seconds*]

DESCRIPTION

The **xload** program displays a periodically updating histogram of the system load average. This program is nothing more than a wrapper around the Athena Load widget.

OPTIONS

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

–scale *integer*	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*	Specifies the frequency in seconds at which **xload** updates its display. Expose events will cause automatic updating. The minimum as well as default time is 5 seconds.

The following standard X Toolkit arguments are commonly used with **xload**.

–bd *color*	Specifies the border color. The default color is black.
–bg *color*	Specifies the background color. The default color is white.
–bw *pixels*	Specifies the width in pixels of the border around the window. The default value is 2.
–fg *color*	Specifies the graph color. The default color is black.
–fn *fontname*	Specifies the font to be used in displaying the name of the host whose load is being monitored. The default is 6x10.
–rv	Indicates that reverse video should be simulated by swapping the foreground and background colors.
–geometry *geometry*	Specifies the prefered size and postion of the window; see X(1).
–display *display*	Specifies the X server to contact; see X(1).
–xrm *resourcestring*	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 *Load* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)	Specifies the width of the load average graph.
height (class **Height**)	Specifies the height of the load average graph.
update (class **Interval**)	Specifies the frequency in seconds at which the load should be redisplayed.
scale (class **Scale**)	Specifies the initial number of ticks on the graph. The default is 1.
minScale (class **Scale**)	Specifies the minimum number of ticks that will be displayed. The default is 1.
foreground (class **Foreground**)	Specifies the color for the graph. Using the class specifies the color for all things that normally would appear in the foreground color. The default is black since the core default for background is white.

label (class **Label**) Specifies the label to use on the graph. The default is the hostname.

font (class **Font**) Specifies the font to be used for the label. The default is "fixed."

reverseVideo (class **ReverseVideo**) Specifies that the foreground and background should be reversed.

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), **mem**(4), Athena Load widget

DIAGNOSTICS

Unable to open display or create window.

Unable to open */dev/kmem*.

Unable to query window for dimensions.

Various X errors.

BUGS

This program requires the ability to open and read */dev/kmem*. On most systems, this requires the suid bit set with root ownership or the sgid bit set and membership in the same group as */dev/kmem*.

Reading */dev/kmem* is inherently non-portable.

Border color has to be explicitly specified when reverse video is used.

NAME

xlogo – X Window System logo

SYNTAX

xlogo [*-toolkitoption*]

DESCRIPTION

The **xlogo** program displays the X Window System logo. This program is nothing more than a wrapper around the Athena Logo widget.

OPTIONS

Xlogo accepts all of the standard X Toolkit command line options, of which the following are commonly used:

–bg *color*	Specifies the color to use for the background of the window. The default is white. A correct color for the background is maroon.
–bd *color*	Specifies the color to use for the border of the window. The default is black.
–bw *number*	Specifies the width in pixels of the border surrounding the window.
–fg *color*	Specifies the color to use for displaying the logo. The default is black. A correct color for the background is silver, which you can approximate with a shade of gray, like #aa9.
–rv	Indicates that reverse video should be simulated by swapping the foreground and background colors.
–geometry *geometry*	Specifies the prefered size and position of the logo window; see **X**(1).
–display *display*	Specifies the X server to contact; see **X**(1).
–xrm *resourcestring*	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 *Logo* widget in the X Toolkit. It understands all of the core resource names and classes as well as:

width (class **Width**)	Specifies the width of the logo.
height (class **Height**)	Specifies the height of the logo.
foreground (class **Foreground**)	Specifies the color for the logo. The default is black since the core default for background is white.
reverseVideo (class **ReverseVideo**)	Specifies that the foreground and background should be reversed.

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), Athena Logo widget

NAME

xlsfonts – server font list displayer for X

SYNTAX

xlsfonts [*-options*] [*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:display*	Specifies the host and server to use.
–l	Indicates that a long listing should be generated for each font.
–m	Indicates that long listings should also print the minimum and maximum bounds of each font.
–C	Indicates that listings should use multiple columns.
–1	Indicates that listings should use a single column.

SEE ALSO

X(1), **Xserver**(1), **xset**(1), **xfd**(1)

ENVIRONMENT

DISPLAY	To get the default host and display to use.

BUGS

Doing "xlsfonts -l" can tie up your server for a very long time. This is really a bug with single-threaded, non-preemptible servers, not with this program.

NAME

xmh – X window interface to the mh Mail Handler

SYNTAX

xmh [-path *mailpath*] [-initial *foldername*] [-flag] [*-toolkitoption*]

DESCRIPTION

The **xmh** program provides a window-oriented front end to the mh Mail Handler. It is designed to take advantage of a large graphical display and pointer. It will not function on an ordinary terminal at all.

Xmh consists of user-interface code only. To actually do things with your mail, it makes calls to the *mh* package.

OPTIONS

-display *host:display*	Specifies the host and server to use.
-path *mailpath*	Specifies the path to the desired mail file. The default is */usr/spool/mail/username*.
-initial *foldername*	Specifies which folder message list should be displayed on startup. The default is "inbox."
-flag	Indicates (if icon is set) that mail is pending or has been received.

INSTALLATION

The current version of **xmh** requires that the user is already set up to use *mh*, version 6. To do so, see if there is a file called *.mh_profile* in your home directory. If you do, check to see if it contains a line that starts with "Current-Folder." If it does, then you've been using version 4 or earlier of *mh*; to convert to version 6, you must remove that line. (Failure to do so causes spurious output to stderr, which can hang **xmh** depending on your setup.)

If you do not already have a *.mh_profile*, you can create one (and everything else you need) by typing "inc" to the shell.

For more information, refer to the *mh* documentation.

RUNNING XMH

Run **xmh** as you would any other X application (e.g., **xterm**). It accepts a command-line display (of the form "-display *host:display*"); the default display is specified in the environment variable DISPLAY.

It is best to run **xmh** while reading this man page so that you see first hand the things being described.

BASIC SCREEN LAYOUT

Xmh starts out with a single screen. There are 6 or 7 areas on the screen:

- A list of your folders. (New users of mh will see only inbox here.)
- A list of the global and folder-oriented commands.
- A list of the messages in one of your folders (initially, this will show the messages in inbox).
- A list of the message-oriented commands.
- A view of one of your messages. (Initially this is blank.)
- A list of commands for the message being viewed.

And, there will possibly be:

- A list of message-sequences defined for this folder. This appears just below the list of messages in this folder. (Message-sequences are discussed below; if you don't know what they are, then you won't have any.)

XMH AND THE TOOLKIT

Xmh uses the X Toolkit. Many of the features described below (scrollbars, buttonboxes, etc.) are actually part of the Toolkit, and are described here only for completeness. For more information, see the Toolkit documentation in your core documentation set.

SCROLLBARS

Some parts of the screen have a vertical area on the left containing a grey bar. This area is a *scrollbar*. They are used whenever the data in a window takes up more space than can be displayed. The grey bar indicates what portion of your data is visible. Thus, if the entire length of the area is grey, then you are looking at all your data. If only the first half is grey, then you are looking at the top half of your data.

You can use the pointer in the scrollbar to change what part of the data is visible. If you click with button 2 (middle mouse button), then the top of the grey area moves to where the pointer is, and the corresponding portion of data is displayed. If you hold down the middle button, you can drag around the grey area. This makes it easy to get to the top of the data: just press and hold the middle mouse button, drag off the top of the scrollbar, and release.

If you click with button 1 (left mouse button), then the data to the right of the pointer scrolls to the top of the window. If you click with pointer button 3 (right mouse button), then the data at the top of the window scrolls down to where the pointer is.

BUTTONBOXES

Any area consisting of many words or short phrases, each enclosed in a box, is called a *buttonbox*. Each box is actually a button that you can press by moving the pointer onto it and pressing pointer button 1. If a given buttonbox has more buttons in it than can fit, it is displayed with a scrollbar, so you can always scroll to the button you want.

ADJUSTING THE RELATIVE SIZES OF AREAS

If you're not satisfied with the size of the various areas on the screen, they can easily be changed. Near the right edge of the border between each region is a black box, called a *grip*. Simply point to that grip with the pointer, press a pointer button, drag up or down, and release. Exactly what happens depends on which pointer button you press.

If you drag with the middle mouse button (pointer button 2), then only that border will move. This mode is simplest to understand, but is probably the least useful.

If you drag with the left mouse button (pointer button 1), then you are adjusting the size of the window above. **Xmh** will attempt to compensate by adjusting some window below it.

If you drag with the right mouse button (pointer button 3), then you are adjusting the size of the window below. **Xmh** will attempt to compensate by adjusting some window above it.

All windows have a mininum and maximum size; you will never be allowed to move a border past the point where it would make a window have an invalid size.

SELECTED FOLDER

The selected folder is whichever foldername is highlighted in the top buttonbox. Note that this is not necessarily the same folder that is being viewed. To change the selected folder, just press on the desired folder button.

GENERAL COMMANDS AND FOLDER COMMANDS

The second buttonbox contains commands of a global nature:

close (or Quit XMH) Exits **xmh**, after first checking that you won't lose any changes.

compose (or Compose Message) Composes a new message. A new window is brought up; for a description of it, see COMPOSITION WINDOWS below.

open (or Open Folder) Display the data in the selected folder. The selected folder also becomes the viewed folder.

openInNew (or Open Folder in New Window) Creates a new screen and displays the selected folder in that screen. Note, however, that you may not display the same folder in more than one screen at a time.

create (or Create Folder) Create a new folder. You will be prompted for a name for the new folder; to enter the name, point the pointer at the blank box provided and type. Hit the Confirm button when finished, or hit Abort to cancel this operation.

delete (or Delete Folder) Destroy the selected folder. You are asked to confirm this action (see CONFIRMATION WINDOWS).

HIGHLIGHTED MESSAGES, SELECTED MESSAGES AND THE CURRENT MESSAGE

It is possible to highlight a set of messages in the list of messages for the viewed folder. To highlight a message, just click on it with pointer button 1. To highlight a range of messages, click on the first one with pointer button 1 and on the last one with pointer button 3.

The selected messages are the same as the highlighted messages, if any. If no messages are highlighted, then the selected messages are considered the same as the current message.

The current message is indicated by a "+" next to the message number. It usually corresponds to the message currently being viewed.

MESSAGE COMMANDS

The third buttonbox (fourth if you have message-sequences displayed) contains commands to deal with messages:

inc (or Incorporate New Mail) Add any new mail received to your inbox folder, and set the current message to be the first new message. (This button is selectable only if "inbox" is the folder being viewed.)

next (or View Next Message) View the first selected message. If no messages are highlighted, view the current message. If the current message is already being viewed, view the first unmarked message after the current message.

prev (or View Previous Message) View the last selected message. If no messages are highlighted, view the current message. If the current message is already being viewed, view the first unmarked message before the current message.

delete (or Mark Deleted) Mark the selected messages for deletion. If no messages are highlighted, then this will automatically display the next unmarked message.

move (or Mark Move) Mark the selected messages to be moved into the current folder. (If the current folder is the same as the viewed folder, this command will just beep.) If no messages are highlighted, then this will automatically display the next unmarked message.

copy (or Mark Copy) Mark the selected messages to be copied into the current folder. (If the current folder is the same as the viewed folder, this command will just beep.)

unmark Remove any of the above three marks from the selected messages.

viewNew (or View in New Window) Create a new window containing only a view of the first selected message.

reply Create a composition window in reply to the first selected message.

forward Create a composition window whose body is initialized to be the contents of the selected messages.

useAsComp (or Use as Composition) Create a composition window whose body is initialized to be this message. Note that any changes you make in the composition are also saved in this message. This function is meant to be used with the "drafts" folder (see COMPOSITION WINDOWS below).

commit (or Commit Changes) Execute any deletions, moves, and copies that have been marked in this folder.

print Print the selected messages. **Xmh** normally prints by invoking the **enscript**(1) command, but you may change the command it uses (see CUSTOMIZING below).

pack (or Pack folder) Renumber the messages in this folder so they start with 1 and increment by 1.

sort (or Sort folder) Sort the messages in this folder in chronological order. As a side effect, this also packs the folder.

rescan (or Force Rescan) Rebuild the list of messages. This can be used whenever you suspect **xmh**'s idea of what messages you have is wrong. (In particular, this is useful if you ever change things using straight mh commands without using **xmh**.)

pick (or Pick Messages) Define a new message-sequence (see MESSAGE-SEQUENCES below).

The following buttons will appear but will be sensitive only if the current folder has any message-sequences defined (see MESSAGE-SEQUENCES below).

openSeq (or Open Sequence) Change the viewed sequence to be the same as the selected sequence.

addToSeq (or Add to Sequence) Add the selected messages to the selected sequence.

removeFromSeq (or Remove from Sequence) Remove the selected messages from the selected sequence.

deleteSeq (or Delete Sequence) Remove the selected sequence entirely. Note the messages themselves are not affected; they simply are no longer grouped together as a message-sequence.

VIEW WINDOWS

The commands in these windows are the same as the message commands by the same name, except instead of affecting the selected messages, they affect the viewed message. In addition, there is the "edit" button, which allows you to edit the message being viewed. While editing, the "edit" button will change to a "save" button which should be pressed to save your edits.

COMPOSITION WINDOWS

Aside from the normal text editing functions, there are three command buttons associated with composition windows:

abort (or Abort Comp) Abort this composition window. If changes have been made, you will be asked to confirm losing them.

send Send this composition. If any errors appear in the message header, you will receive a mail message containing this composition and a description of the error.

save Save this composition in your drafts folder; (if you do not have a folder named "drafts," one will be created) then you can safely close the composition. At some future date, you can continue working on the composition by opening your drafts folder, selecting the message, and using the "Use as Composition" command.

TEXT EDITING COMMANDS

All of the text editing commands are actually defined by the *Text* widget in the X Toolkit. The commands may be bound to different keys than the defaults described below through the standard X Toolkit key rebinding mechanisms. See the X Toolkit and Athena Widgets documentation for more details.

Whenever you are asked to enter any text, you will be using a standard text editing interface. Various control and meta keystroke combinations are bound to a somewhat Emacs-like set of commands. In addition, the pointer buttons may be used to select a portion of text or to move the insertion point in the text. Pressing pointer button 1 causes the insertion point to move to the pointer. Double-clicking button 1 selects a word, triple-clicking selects a paragraph, and quadruple-clicking selects everything. Any selection may be extended in either direction by using pointer button 3.

In the following, a *line* refers to one displayed row of characters in the window. A *paragraph* refers to the text between carriage returns. Text within a paragraph is broken into lines based on the current width of the window.

The following keystroke combinations are defined:

Control-A Move to the beginning of the current line.

Control-B, Control-H, Backspace Move backward one character.

Control-D Delete the next character.

Control-E Move to the end of the current line.

Control-F Move forward one character.

Control-J, LineFeed Create a new paragraph with the same indentation as the previous one.

Control-K Kill the rest of this line.

Control-L Repaint this window.

Control-M, Return Create a new paragraph.

Control-N	Move down to the next line.
Control-O	Break this paragraph into two.
Control-P	Move up to the previous line.
Control-V	Move down to the next screenfull of text.
Control-W	Kill the selected text.
Control-Y	Insert the last killed text.
Control-Z	Scroll the text one line up.
Meta-<	Move to the beginning of the document.
Meta->	Move to the end of the document.
Meta-[	Move backward one paragraph.
Meta-]	Move forward one paragraph.
Meta-B	Move backward one word.
Meta-D	Kill the next word.
Meta-F	Move forward one word.
Meta-H, Meta-Delete	Kill the previous word.
Meta-I	Insert a file. If any text is selected, use the selected text as the filename. Otherwise, a box will appear in which you can type the desired filename.
Meta-V	Move up to the previous screenfull of text.
Meta-Y	Insert the last selected text here. Note that this can be text selected in some other text subwindow. Also, if you select some text in an **xterm** window, it may be inserted in an **xmh** window with this command. This action is equivalent to pressing pointer button 2.
Meta-Z	Scroll the text one line down.
Delete	Delete the previous character.

CONFIRMATION WINDOWS

Whenever you press a button that may cause you to lose some work or is otherwise dangerous, a window will appear asking you to confirm the action. This window will contain an "Abort" button and a "Confirm" button. Pressing the "Abort" button cancels the operation, and pressing the "Confirm" button will proceed with the operation. (A very handy shortcut exists: if you press the offending button again, it will be interpreted as a "Confirm." If you press any other command button, it will be interpreted as an "Abort.")

MESSAGE-SEQUENCES

An mh message sequence is just a set of messages associated with some name. They are local to a particular folder; two different folders can have sequences with the same name. In all folders, the sequence "all" is predefined; it consists of the set of all messages in that folder. (The sequence "cur" is also usually defined for every folder; it consists of only the current message. **Xmh** hides "cur" from the user, instead placing a "+" by the current message. Also, **xmh** does not support the "unseen" sequence, so that one is also hidden from the user.)

The message sequences for a folder are displayed as buttons containing the names of the sequences (including one for "all"). The table of contents (aka "toc") is at any one time displaying one message sequence. This is called the "viewed sequence"; if it's not "all," its name will be displayed in the title bar

just after the folder name. Also, at any time one of the sequence buttons will be highlighted. This is called the "selected sequence." Note that the viewed sequence and the selected sequence are not necessarily the same. (This all pretty much corresponds to the way the folder buttons work.)

The **Open Sequence**, **Add to Sequence**, **Remove from Sequence**, and **Delete Sequence** buttons are active only if the viewed folder contains message-sequences.

Note that none of the above actually effect whether a message is in the folder. Remember that a sequence is a set of messages within the folder; the above operations just affect what messages are in that set.

To create a new sequence, press the "Pick" button. A new window will appear, with lots of places to enter text. Basically, you can describe the sequence's initial set of messages based on characteristics of the message. Thus, you can define a sequence to be all the messages that were from a particular person, or with a particular subject, and so on. You can also connect things up with boolean operators, so you can select all things from "weissman" with the subject "xmh."

Hopefully, the layout is fairly obvious. The simplest cases are the easiest: just point to the proper field and type. If you enter in more than one field, it will only select messages which match all non-empty fields.

The more complicated cases arise when you want things that match one field or another, but not necessarily both. That's what all the "or" buttons are for. If you want all things with the subject "xmh" or "xterm," just press the "or" button next to the "Subject:" field. Another box will appear where you can enter another subject.

If you want all things either from "weissman" or with subject "xmh," but not necessarily both, select the "-Or-" button. This will essentially double the size of the form. You can then enter "weissman" in a from: box on the top half, and "xmh" in a subject: box on the lower part.

If you ever select the "Skip" button, then only those messages that *don't* match the fields on that row are included.

Finally, in the bottom part of the window will appear several more boxes. One is the name of the sequence you're defining. (It defaults to the name of the selected sequence when "Pick" was pressed, or to "temp" if "all" was the selected sequence.) Another box defines which sequence to look through for potential members of this sequence; it defaults to the viewed sequence when "Pick" was pressed.

Two more boxes define a date range; only messages within that date range will be considered. These dates must be entered in 822-style format: each date is of the form "dd mmm yy hh:mm:ss zzz," where dd is a one or two digit day of the month, mmm is the three-letter abbreviation for a month, and yy is a year. The remaining fields are optional: hh, mm, and ss specify a time of day, and zzz selects a time zone. Note that if the time is left out, it defaults to midnight; thus, if you select a range of "7 nov 86" - "8 nov 86," you will only get messages from the 7th, as all messages on the 8th will have arrived after midnight.

"Date field" specifies which date field in the header to look at for this date range; it probably won't be useful to anyone. If the sequence you're defining already exists, you can optionally merge the old set with the new; that's what the "Yes" and "No" buttons are all about. Finally, you can "OK" the whole thing, or "Cancel" it.

In general, most people will rarely use these features. However, it's nice to occasionally use "Pick" to find some messages, look through them, and then hit "Delete Sequence" to put things back in their original state.

CUSTOMIZING XMH

As with all standard X applications, **xmh** may be customized through entries in the resource manager. The following resource manager entries are defined: [Note: the entry names must be entered in either all lower-case, or in the exact case shown below.]

BackGround Background color. Currently, this will effect only buttons. (Default is white.)

ButtonFont What font to use for button names. (Default is timrom10.)

CheckNewMail If True, **xmh** will check at regular intervals to see if new mail has arrived for any of the folders. A visual indication will be given if new mail is waiting to be retrieved. (Default is True.)

CompButtonLines How many rows of buttons to display under a composition. (Default is 1.)

CompFont What font to use when composing a message. (Default is 6x13.)

CompGeometry Initial geometry for windows containing compositions.

CompLines How many lines of a composition to display. (Default is 20.)

ConfirmFont What font to use for confirmation windows. (Default is timrom10b.)

FolderButtonLines How many rows of folder command buttons to display. (Default is 1.)

FolderLines How many rows of foldername buttons to display. (Default is 1.)

ForeGround Foreground color. Currently, this will effect only title bars and buttons. (Default is black.)

Geometry Default geometry to use. (Default is none.)

HideBoringHeaders If on, then Bmh will attempt to skip uninteresting header lines within messages by scrolling them off. (Default is on.)

InitialFolder Which folder to display on startup. May also be set with the command-line option **-initial**. (Default is inbox.)

InitialIncFile The file name of your incoming mail drop. **Xmh** tries to construct a filename for the "inc -file" command, but in some installations (e.g., those using the Post Office Protocol) no file is appropriate. In this case, **InitialIncFile** should be specified as the empty string, and *inc* will be invoked without a -file argument.

LabelFont What font to use for the title bars. (Default is timrom10i.)

MailPath The full path prefix for locating your mail folders. May also be set with the command-line option, **-path**. (Default is the Path component in *$HOME/.mh_profile*, or $HOME/Mail if none.)

MailWaitingFlag If True, **xmh** will attempt to set an indication in it's icon when new mail is waiting to be retrieved. If this option is True, then CheckNewMail is assumed to be True as well. The **-flag** command line option is a quick way to turn MailWaitingFlag on.

MhPath What directory in which to find the mh commands. If a command isn't found here, then the directories in the user's path are searched. (Default is */usr/local/mh6*.)

PickGeometry Initial geometry for pick windows.

PickEntryFont What font to use for user text fields in pick windows. (Default is timrom10.)

PickTextFont What font to use for static text fields in pick windows. (Default is timrom10.)

PrintCommand	What sh command to execute to print a message. Note that stdout and stderr must be specifically redirected! If a message or range of messages is selected for printing, the full file paths of each message file is appended to the specified print command. (Default is enscript >/dev/null 2>/dev/null.)
TempDir	Directory for **xmh** to store temporary directories. For privacy, a user might want to change this to a private directory. (Default is */tmp*.)
TocButtonLines	How many rows of message command buttons to display. (Default is 1.)
TocFont	What font to use for a folder's table of contents. (Default is 6x13.)
TocGeometry	Initial geometry for master **xmh** windows.
TocLines	How many messages to display in a folder's table of contents. (Default is 10.)
TocWidth	How many characters to generate for each message in a folder's table of contents. (Default is 100. Use 80 if you plan to use *mhe* a lot.)
ViewButtonLines	How many rows of buttons to display under a view of a message. (Default is 1.)
ViewFont	What font to use for a view of a message. (Default is 6x13.)
ViewGeometry	Initial geometry for windows showing only a view of a message.
ViewLines	How many lines of a message to display. (Default is 20.)

If **TocGeometry, ViewGeometry, CompGeometry,** or **PickGeometry** are not specified, then the value of Geometry is used instead. If the resulting height is not specified (e.g., "", "=500," "+0-0)", then the default height is calculated from the fonts and line counts specified above. If the width is not specified (e.g., "", "=x300," "-0+0"), then half of the display width is used. If unspecified, the height of a pick window defaults to half the height of the display.

Any of these options may also be specified on the command line by using the standard X Toolkit resource specification mechanism. Thus, to run **xmh** showing all message headers,

% xmh -xrm '*HideBoringHeaders:off'

FILES

~/Mail
~/.mh_profile

SEE ALSO

X(1), **xrdb**(1), X Toolkit, **mh**(1) - the mh Mail Handler, **enscript**(1)

BUGS

Printing support is minimal.

Keyboard shortcuts for commands would be nice.

Should handle the "unseen" message-sequence.

Should determine by itself if the user hasn't used *mh* before, and offer to set things up for him or her.

Still a few commands missing (rename folder, remail message).

Needs sub-folder support.

NAME

xmodmap, xprkbd – keyboard modifier utilities for X

SYNTAX

xmodmap [*-options*] [*filename*]
xprkbd [-display *display*]

DESCRIPTION

Xmodmap is a utility for displaying and altering the X keyboard modifier map and keysym table on the specified display and host. It is intended to be run from a user's X startup script to setup the keyboard according to personal tastes.

With no arguments, **xmodmap** displays the current map.

Xprkbd prints the following on the standard output: a table of the keycodes, the keysym code, and the keynames for the keyboard on the appropriate X server.

OPTIONS

Both programs accept the following option:

–display *display* Specifies the display to use; see **X**(1).

The **xmodmap** program also accepts:

–help 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 Indicates that a help message describing the expression grammar used in files and with -e expressions should be printed on the standard error.

–verbose Indicates that **xmodmap** should print logging information as it parses its input.

–quiet Turns off the verbose logging. This is the default.

–n 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* Specifies an expression to be executed. Any number of expressions may be specified from the command line.

–p Indicates that the current modifier map should be printed on the standard output.

– Indicates 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 *.keymap.km.*

For compatibility with an older version, **xmodmap** also accepts the following obsolete single letter options:

-[SLC12345] Indicates that all current keys for the Shift, Lock, Control, or Mod modifier sets should be removed from the modifier map. These are equivalent to **clear** expressions.

-[slc] *keysym* Specifies a **keysym** to be removed from the Shift, Lock, or Control modifier sets. These are equivalent to **remove** expressions.

+[slc12345] *keysym* Specifies a **keysym** to be added to the Shift, Lock, or Control modifier sets. These are equivalent to **add** expressions.

EXPRESSION GRAMMAR

The **xmodmap** program reads a list of expressions and converts them into appropriate calls to the following Xlib routines: **XChangeKeyboardMapping**(3X), **XInsertModifiermapEntry**(3X), and **XDeleteModifiermapEntry**(3X). Allowable expressions include:

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 names are: Shift, Lock, Control, Mod1, Mod2, Mod3, Mod4 and Mod5 (case does not matter in modifier names, although it does matter for all other names). For example, "clear Lock" will remove all 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 below).

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.

Lines that begin with an exclamation mark (!) 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

To make the backspace key generate a delete, use

```
% xmodmap -e "keysym BackSpace = Delete"
```

To swap the left control and caps lock keys, use:

```
!
! 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
```

As a more complicated example of swapping the left control and caps lock keys, you could use:

```
!
! On the HP, the following keycodes have key caps as listed:
!
!   101  Backspace
```

```
!    55 Caps
!    14 Ctrl
!    15 Break/Reset
!    86 Stop
!    89 F5
!
! Using "keycode" over "keysym" you can rerun the file to
! fix up your keyboard.
!
! This sets the backspace key to generate Delete, flushes all caps lock
! bindings, assigns a control key to what used to be the caps lock key,
! makes the F1 generate ESC, and assigns
! the Break/Reset key to be a shift lock.

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), **xprkbd**(1), **make**(1), **XChangeKeyboardMapping**(3X), **XInsertModifiermapEntry**(3X), **XDeleteModifiermapEntry**(3X)

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.

NAME

xpr – print an X window dump

SYNTAX

xpr [-scale *scale*] [-height *inches*] [-width *inches*] [-left *inches*] [-top *inches*] [-header *string*] [-trailer *string*] [-landscape] [-portrait] [-rv] [-compact] [-output *filename*] [-append *filename*] [-noff] [-split *n*] [-device *dev] [filename]*

DESCRIPTION

Xpr takes as input a window dump file produced by **xwd**(1) and formats it for output on the LN03, LA100, PostScript printers, or IBM PP3812 page printer. If no file argument is given, the standard input is used. By default, **xpr** prints the largest possible representation of the window on the output page. Options allow you 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.

OPTIONS

–scale *scale*
: Affects the size of the window on the page. The LN03 and PostScript printers are able to translate each bit in a window pixel map into a grid of a specified size. For example, each bit might translate into a 3x3 grid. This would be specified by **–scale** *3*. By default a window is printed with the largest scale that will fit onto the page for the specified orientation.

–height *inches*
: Specifies the maximum height of the window on the page.

–width *inches*
: Specifies the maximum width of the window.

–left *inches*
: Specifies the left margin in inches. Fractions are allowed. By default the window is centered in the page.

–top *inches*
: Specifies the top margin for the picture in inches. Fractions are allowed.

–header *header*
: Specifies a header string to be printed above the window.

–trailer *trailer*
: Specifies a trailer string to be printed below the window.

–landscape
: Forces the window to be printed in landscape mode. By default a window is printed such that its longest side follows the long side of the paper.

–portrait
: Forces the window to be printed in portrait mode. By default a window is printed such that its longest side follows the long side of the paper.

–rv
: Forces the window to be printed in reverse video.

–compact
: Uses simple run-length encoding for compact representation of windows with lots of white pixels.

–output *filename*
: Specifies an output file name. If this option is not specified, standard output is used.

–append *filename*
: Specifies a filename previously produced by **xpr** to which the window is to be appended.

–noff
: When specified in conjunction with **–append**, the window appears on the same page as the previous window.

–split *n*
: Allows you to split a window onto several pages. This might be necessary for very large windows that would otherwise cause the printer to overload and print the page in an obscure manner.

–device *device*
: Specifies the device on which the file will be printed. Currently only the LN03 (-device ln03), LA100 (-device la100), PostScript printers (-device ps), and IBM PP3812 (-device pp) are supported. **-device lw** (LaserWriter) is equivalent to

-device ps and is provided only for backwards compatibility.

SEE ALSO

xwd(1), **xdpr**(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 halftones 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 rework the application to produce a less complex picture.

Xpr provides some support for the LA100. However, there are several limitations on its use: The picture will always be printed in portrait mode, there is no scaling, and the aspect ratio will be slightly off.

Support for PostScript output currently cannot handle the **-append**, **-noff**, or **-split** options.

The **-compact** option is *only* supported for PostScript output. It compresses white space but not black space, so it is not useful for reverse-video windows.

NAME

xmodmap, xprkbd – keyboard modifier utilities for X

SYNTAX

xmodmap [*-options*] [*filename*]

xprkbd [-display *display*]

DESCRIPTION

Xmodmap is a utility for displaying and altering the X keyboard modifier map and keysym table on the specified display and host. It is intended to be run from a user's X startup script to setup the keyboard according to personal tastes.

With no arguments, **xmodmap** displays the current map.

Xprkbd prints on the standard output a table of the keycodes, the keysym code, and the keynames for the keyboard on the appropriate X server.

OPTIONS

Both programs accept the following option:

–display *display* Specifies the display to use; see X(1).

The **xmodmap** program also accepts:

–help 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 Indicates that a help message describing the expression grammar used in files and with **-e** expressions should be printed on the standard error.

–verbose Indicates that **xmodmap** should print logging information as it parses its input.

–quiet Turns off the verbose logging. This is the default.

–n 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* Specifies an expression to be executed. Any number of expressions may be specified from the command line.

–p Indicates that the current modifier 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 *.keymap.km.*

For compatibility with an older version, **xmodmap** also accepts the following obsolete single letter options:

-[SLC12345] Indicates that all current keys for the Shift, Lock, Control, or Mod modifier sets should be removed from the modifier map. These are equivalent to **clear** expressions.

-[slc] *keysym* Specifies a *keysym* to be removed from the Shift, Lock, or Control modifier sets. These are equivalent to **remove** expressions.

+[slc12345] *keysym* Specifies a *keysym* to be added to the Shift, Lock, or Control modifier sets. These are equivalent to **add** expressions.

EXPRESSION GRAMMAR

The **xmodmap** program reads a list of expressions and converts them into appropriate calls to the following *Xlib* routines: **XChangeKeyboardMapping**(3X), **XInsertModifiermapEntry**(3X) and **XDeleteModifiermapEntry**(3X). Allowable expressions include:

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.

Lines that begin with an exclamation mark (!) 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

To make the backspace key generate a delete instead, use

```
% xmodmap -e "keysym BackSpace = Delete"
```

To swap the left control and caps lock keys you could use:

```
!
! 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
```

As a more complicated example, the following is what the author uses:

```
!
! On the HP, the following keycodes have key caps as listed:
!
!   101  Backspace
!    55  Caps
!    14  Ctrl
```

```
!    15 Break/Reset
!    86 Stop
!    89 F5
!
! I prefer using "keycode" over "keysym" so that I can rerun the file to
! fix up my keyboard.
!
! This sets the backspace key to generate Delete, flushes all caps lock
! bindings, assigned a control key to what used to be the caps lock key,
! makes the F1 generate ESC, and makes the Break/Reset key be a shift lock.

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), **make**(1), **xmodmap**(1), **XChangeKeyboardMapping**(3X), **XInsertModifiermapEntry**(3X), XDeleteModifiermapEntry(3X)

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.

NAME

xprop – property displayer for X

SYNTAX

xprop [-help] [-grammar] [-id *id*] [-root] [-name *name*] [-font *font*] [-display *display*] [-len *n*] [-notype] [-fs *file*] [-f *atom format* [*dformat*]]* [*format* [*dformat*] *atom*]*

SUMMARY

The **xprop** 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 Prints out a summary of command line options.

-grammar Prints out a detailed grammar for all command line options.

-id *id* Allows you 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* Allows you 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* Allows you to specify that the properties of font *font* should be displayed.

-root Specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

-display *display* 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.

-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 you are 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 may be used.

Other than the above four arguments, the **-help** argument for obtaining help, and the **-grammar** argument for listing the full grammar for the command line, all 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 you dislike the standard display format. New users 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 orginally 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?) 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 a 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 manor 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 $, ?, \, or (preceed it by a \ (i.e., 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* will display character number *o*

where *o* is an octal number.

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 corrsponding *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 is 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 corrsponding *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 refered 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) Xlib manual (on-line version)

NAME

xrdb – X server resource database utility

SYNTAX

xrdb [*-option*] [*filename*]

DESCRIPTION

Xrdb is used to get or set the contents of the RESOURCE_MANAGER property on the root window of screen 0. You would normally run this program from your X startup file.

The resource manager (used by the Xlib routine **XGetDefault**(3X) and the X Toolkit) uses the RESOURCE_MANAGER property to get user preferences about color, fonts, and so on for applications. Having this information in the server (where it is available to all clients) instead of on disk, solves the problem in previous versions of X that required you to maintain *defaults* files on every machine that you might use. It also allows for dynamic changing of defaults without editing files.

For compatibility, if there is no RESOURCE_MANAGER property defined (either because **xrdb** was not run or if the property was removed), the resource manager will look for a file called *.Xdefaults* in your home directory.

The *filename* (or the standard input if "-" or no input file is given) is optionally passed through the C preprocessor with the following symbols defined, based on the capabilities of the server being used:

HOST=hostname	The hostname portion of the display to which you are connected.
WIDTH=num	The width of the screen in pixels.
HEIGHT=num	The height of the screen in pixels.
X_RESOLUTION=num	The x resolution of the screen in pixels per meter.
Y_RESOLUTION=num	The y resolution of the screen in pixels per meter.
PLANES=num	The number of bit planes for the default visual.
CLASS=visualclass	One of StaticGray, GrayScale, StaticColor, PsuedoColor, TrueColor, DirectColor.
COLOR	Only defined if the default visual's type is one of the color options.

Lines that begin with an exclamation mark (!) are ignored and may be used as comments.

OPTIONS

–help	Causes a brief description of the allowable options and parameters to be printed.
–display *display*	Specifies the X server to be used; see X(1).
–cpp *filename*	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	Indicates that **xrdb** should not run the input file through a preprocessor before loading it into the RESOURCE_MANAGER property.
–symbols	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	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 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 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 change the contents of the RESOURCE_MANAGER property directly from a terminal or from a shell script.

–remove Indicates that the RESOURCE_MANAGER property should be removed from its window.

–edit *filename* 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* Specifies a suffix to be appended to the filename used with **–edit** to generate a backup file.

–D*name[=value]* Passes through to the preprocessor and is used to define symbols for use with conditionals such as *#ifdef*.

–U*name* Passes through to the preprocessor and is used to remove any definitions of this symbol.

–I*directory* Passes 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.

NAME

xrefresh – refresh all or part of an X screen

SYNTAX

xrefresh [*-option*]

DESCRIPTION

Xrefresh is a simple X program that causes all or part of your screen to be repainted. **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.

OPTIONS

–white	Specifies a white background. The screen just appears to flash quickly, and then repaint.
–black	Specifies 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*	Specifies a solid color background. Try green.
–root	Specifies the root window background.
–none	Repaints all of the windows. (This is the default.)
–geometry *WxH+X+Y*	Specifies the portion of the screen to be repainted. This superceeds the old style **=WxH+X+Y**.
–display *display*	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), **XGetDefault**(3X)

BUGS

It should have just one default type for the background.

NAME

xset – user preference utility for X

SYNTAX

xset [-display *display*] [-b] [b on/off] [b [*volume* [*pitch* [*duration*]]] [-c] [c on/off] [c [*volume*]] [fp *path*[,*path*[,...]]] [fp default] [[-]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 sets various user preference options of the display.

OPTIONS

–display *display* Specifies which server to use; see X(1).

b Controls the bell volume, pitch, and duration. This option accepts up to three numerical parameters, a preceding dash (-), or an "on/off" flag. If no parameters are given, or the "on" flag is used, the system defaults will be used. If the dash or "off" are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.

c Controls the key click. Takes an optional value, a preceding dash (-), or an "on/off" flag. If no parameter or the "on" flag is given, the system defaults are used. If the dash or "off" flag is used, keyclick is 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 sets the volume to the nearest value that the hardware can support.

fp Sets the font path. It must be followed by a comma-separated list of directories or the flag "default." The indicated path is used to find fonts for clients. To restore the default font path, use **fp default.**

led 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 is 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 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 is interpreted as the acceleration. If no parameters or the flag "default" is used, the system defaults will be set.

p Controls the pixel color values. The parameters are the colormap 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 Controls the autorepeat. If a preceding dash or the "off" flag is used, autorepeat is disabled. If no parameters or the "on" flag is used, autorepeat is enabled.

s Allows you to 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 is 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 is used for the length.

q Provides you information on the current settings.

These settings are 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)

NAME

xsetroot – root window parameter setting utility for X

SYNTAX

xsetroot [-help] [-def] [-display *display*] [-cursor *cursorfile maskfile*] [-bitmap *filename*] [-mod *x y*] [-gray] [-grey] [-fg *color*] [-bg *color*] [-rv] [-solid *color*] [-name *string*]

DESCRIPTION

The **xsetroot** program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with **xsetroot** until you find a personalized look that you like, then put the **xsetroot** command that produces it into your X startup file. If no options are specified, or if **-def** is specified, the window is reset to its default state. The **-def** option can be specified along with other options and only the non-specified characteristics will be reset to the default state.

Only one of the background color/tiling changing options (**-solid, -gray, -grey, -bitmap,** and **-mod**) may be specified at a time.

OPTIONS

-help	Prints a usage message and exits.
-def	Resets unspecified attributes to their default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)
-cursor *cursorfile maskfile*	Changes the cursor to whatever you want when the cursor is outside of any window. Cursor and mask files are bitmaps (little pictures), and can be made with the **bitmap**(1) program. You probably want the mask file to be all black until you get used to the way masks work.
-bitmap *filename*	Uses 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*	Creates 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	Makes the entire background gray. (Easier on the eyes.)
-grey	Makes the entire background grey.
-fg *color*	Uses *color* as the foreground color when setting attributes.
-bg *color*	Uses *color* as the background color when setting attributes.
-rv	Exchanges the foreground and background colors. Normally the foreground color is black and the background color is white.
-solid *color*	Sets the window color to *color*.
-name *string*	Sets 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), **bitmap**(1)

NAME

xterm – terminal emulator for X

SYNTAX

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** uses 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 are restricted to the largest box with a 4014's aspect ratio that fits in the window. This box is located in the upper left area of the window.

Although both windows may be displayed at the same time, one of them is considered the active window for receiving keyboard input and terminal output. This is the window that contains the text cursor and whose border highlights whenever the pointer is in either window. The active window can be chosen through escape sequences, the Modes menu in the VT102 window, and the Tektronix menu in the 4014 window.

OPTIONS

The **xterm** terminal emulator accepts all of the standard X Toolkit command line options along with the additional options listed below (if the option begins with a "+" instead of a "–", the option is restored to its default value):

–132 Causes the DECCOLM escape sequence to be recognized, and the **xterm** window will resize appropriately. Normally, the VT102 DECCOLM escape sequence that switches between 80 and 132 column mode is ignored.

–b *number* 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.

–cr *color* Specifies the color to use for text cursor. The default is to use the same foreground color that is used for text.

–cu 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 Indicates that **xterm** should not work around the **curses**(3X) bug mentioned above.

–e *program [arguments]* Specifies the program (and its command line arguments) to be run in the **xterm** window. The default is to start the user's shell. **This must be the last option on the command line.**

–fb *font* Specifies a font to be used when displaying bold text. This font must be the same height and width as the normal font. If only one of the normal or bold fonts is specified, it will be used as the normal font and the bold font will be produced by overstriking this font. The default bold font is vtbold.

–j 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 makes **xterm** run 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 Indicates that **xterm** should not do jump scrolling.

–l 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 Indicates that **xterm** should not do logging.

–lf *filename* 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 Indicates that the shell started in the **xterm** window is to 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 Indicates that the shell started should not be a login shell (i.e., it will be normal subshell).

–mb 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 Indicates that the margin bell should not be rung.

–ms *color* Specifies the color to be used for the cursor. The default is to use the foreground color.

–nb *number* 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 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 Indicates that reverse-wraparound should not be allowed.

–s 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 Indicates that **xterm** should scroll synchronously.

–sb 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 Indicates that a scrollbar should not be displayed.

–si 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 Indicates that output to a window should cause it to scroll to the bottom.

–sk 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

postion at the bottom of the scroll region.

+sk Indicates that pressing a key while using the scrollbar should not cause the window to be repositioned.

–sl *number* Specifies the number of lines to save that have been scrolled off the top of the screen. The default is 64.

–t Indicates that **xterm** should start in Tektronix mode, rather than in VT102 mode. Switching between the two windows is done using the Modes menu.

+t Indicates that **xterm** should start in VT102 mode.

–vb Indicates that a visual bell is prefered over an audible one. Instead of ringing the terminal bell whenever a Control-G is received, the window will be flashed.

+vb Indicates that a visual bell should not be used.

–C Indicates that this window should receive console output. This is not supported on all systems.

–L Indicates that **xterm** was started by **init**. In this mode, **xterm** does not try to allocate a new pseudoterminal as **init** has already done so. In addition, the system program *getty* is run instead of the user's shell. **This option should never be used when starting terminal windows.**

–S*ccn* Specifies the last two letters of the name of a pseudoterminal to use in slave mode. This allows **xterm** to be used as an input and output channel for an existing program and is sometimes used in specialized applications.

The following command line arguments are provided for compatibility with older versions. They may not be supported in the next release as the X Toolkit provides standard options that accomplish the same task.

%geom Specifies the prefered size and position of the Tektronix window. It is shorthand for specifying the **tekGeometry* resource.

#geom Specifies the prefered position of the icon window. It is shorthand for specifying the **iconGeometry* resource.

–T *string* Specifies the title for **xterm**'s windows. It is equivalent to **-title**.

–n*string* Specifies the icon name for **xterm**'s windows. It is shorthand for specifying the **iconName* resource.

–r Indicates that reverse video should be simulated by swapping the foreground and background colors. It is equivalent to **-reversevideo** or **-rv**.

–w *number* 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* Specifies the color to use for the background of the window. The default is white.

–bd *color* Specifies the color to use for the border of the window. The default is black.

–bw *number* Specifies the width in pixels of the border surrounding the window.

–fg *color* Specifies the color to use for displaying text. The default is black.

–fn *font* Specifies the font to be used for displaying normal text. The default is vtsingle.

–name *name* Specifies the application name under which resources are to be obtained,

rather than the default executable file name.

–rv Indicates that reverse video should be simulated by swapping the foreground and background colors.

–geometry *geometry* Specifies the prefered size and position of the VT102 window (see X(1)).

–display *display* Specifies the X server to contact (see X(1)).

–xrm *resourcestring* Specifies a resource string to be used. This is especially useful for setting resources that do not have separate command line options.

X DEFAULTS

The program understands all of the core X Toolkit resource names and classes as well as:

name (class **Name**) Specifies the name of this instance of the program. The default is xterm.

iconGeometry (class **IconGeometry**)
Specifies the prefered size and position of the application when iconified. It is not necessarily obeyed by all window managers.

title (class **Title**) Specifies a string that may be used by the window manager when displaying this application.

The following resources are specified as part of the vt100 widget (class VT100):

font (class **Font**) Specifies the name of the normal font. The default is vtsingle.

boldFont (class **Font**) Specifies the name of the bold font. The default is vtbold.

c132 (class **C132**) Specifies whether or not the VT102 DECCOLM escape sequence should be honored. The default is false.

curses (class **Curses**) Specifies whether or not the last column bug in cursor should be worked around. The default is false.

background (class **Background**)
Specifies the color to use for the background of the window. The default is white.

foreground (class **Foreground**)
Specifies the color to use for displaying text in the window. Setting the class name instead of the instance name is an easy way to have everything that would normally appear in the text color change color. The default is black.

cursorColor (class **Foreground**)
Specifies the color to use for the text cursor. The default is black.

geometry (class **Geometry**)
Specifies the prefered size and position of the VT102 window.

tekGeometry (class **Geometry**)
Specifies the prefered size and position of the Tektronix window.

internalBorder (class **BorderWidth**)
Specifies the number of pixels between the characters and the window border. The default is 2.

jumpScroll (class **JumpScroll**)
Specifies whether or not jump scroll should be used. The default is false.

logFile (class **Logfile**) Specifies the name of the file to which a terminal session is logged. The default is **XtermLog.***XXXXX* (where *XXXXX* is the process id of **xterm**).

logging (class **Logging**) Specifies whether or not a terminal session should be logged. The default is

false.

logInhibit (class **LogInhibit**)
Specifies whether or not terminal session logging should be inhibited. The default is false.

loginShell (class **LoginShell**)
Specifies whether or not the shell to be run in the window should be started as a login shell. The default is false.

marginBell (class **MarginBell**)
Specifies whether or not the bell should be run when the user types near the right margin. The default is false.

multiScroll (class **MultiScroll**)
Specifies whether or not asynchronous scrolling is allowed. The default is false.

nMarginBell (class **Column**)
Specifies the number of characters from the right margin at which the margin bell should be run, when enabled.

pointerColor (class **Foreground**)
Specifies the color of the pointer. The default is black.

pointerShape (class **Cursor**)
Specifies the name of the shape of the pointer. The default is xterm.

reverseVideo (class **ReverseVideo**)
Specifies whether or not reverse video should be simulated. The default is false.

reverseWrap (class **ReverseWrap**)
Specifies whether or not reverse-wraparound should be enabled. The default is false.

saveLines (class **SaveLines**)
Specifies the number of lines to save beyond the top of the screen when a scrollbar is turned on. The default is 64.

scrollBar (class **ScrollBar**) Specifies whether or not the scrollbar should be displayed. The default is false.

scrollInput (class **ScrollCond**)
Specifies whether or not output to the terminal should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is true.

scrollKey (class **ScrollCond**)
Specifies whether or not pressing a key should automatically cause the scrollbar to go to the bottom of the scrolling region. The default is false.

signalInhibit (class **SignalInhibit**)
Specifies whether or not the entries in the xterm X11 menu for sending signals to **xterm** should be disallowed. The default is false.

tekInhibit (class **TekInhibit**)
Specifies whether or not Tektronix mode should be disallowed. The default is false.

tekStartup (class **TekStartup**)
Specifies whether or not **xterm** should start up in Tektronix mode. The default is false.

visualBell (class **VisualBell**)
Specifies whether or not a visible bell (i.e., flashing) should be used instead of an audible bell when Control-G is received. The default is false.

The following resources are specified as part of the tek4014 widget (class Tek4014):

width (class **Width**) Specifies the width of the Tektronix window in pixels.

height (class **Height**) Specifies the height of the Tektronix window in pixels.

The following resources are specified as part of the menu widget:

menuBorder (class **MenuBorder**)
Specifies the size in pixels of the border surrounding menus. The default is 2.

menuFont (class **Font**) Specifies the name of the font to use for displaying menu items.

menuPad (class **MenuPad**)
Specifies the number of pixels between menu items and the menu border. The default is 3.

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; **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.

The Tektronix 4014 emulation supports four different font sizes and five different lines types. 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 left mouse button is used to save text into the cut buffer. Move the cursor to the 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 saved in the global cut buffer when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection.

The middle mouse button types (pastes) the text from the cut buffer, inserting it as keyboard input.

The right mouse button extends the current selection. (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 treated as if they 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 the left mouse button with the pointer in the scroll region moves the adjacent line to the top of the display window.

Clicking the right mouse button moves the top line of the display window down to the pointer position.

Clicking the middle mouse button moves the display to a position in the saved text that corresponds to the pointer's position in the scrollbar.

Unlike the VT102 window, the Tektronix window does not allow the copying of text. It does allow Tektronix GIN mode, and in this mode the cursor will change from an arrow to a cross. Pressing any key will send that key and the current coordinate of the cross cursor. Pressing the left, middle, or right mouse button 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 distinquish a pointer button from a key, the high bit of the character is set (but this bit is normally stripped unless the terminal mode is RAW; see **tty**(4) for details).

MENUS

Xterm has three different menus, named **xterm**, **Modes**, and **Tektronix.** Each menu pops up under the correct combinations of key and button presses. Most menus are divided into two sections, separated by a horizontal line. The top portion contains various modes that can be altered. A check mark appears next to a mode that is currently active. Selecting one of these modes toggles its state. The bottom portion of the menu consists of command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the control key and pointer button one are pressed in a window. The modes section contains items that apply to both the VT102 and Tektronix windows. Notable entries in the command section of the menu are **Continue**, **Suspend**, **Interrupt**, **Hangup**, **Terminate**, and **Kill** which sends the SIGCONT, SIGTSTP, SIGINT, SIGHUP, SIGTERM, and SIGKILL signals, respectively, to the process group of the process running under **xterm** (usually the shell). The **Continue** function is especially useful if the user has accidentally typed CTRL-Z, suspending the process.

The **Modes** menu sets various modes in the VT102 emulation, and is popped up when the control key and pointer button two are pressed in the VT102 window. In the command section of this menu, the soft reset entry will reset scroll regions. This can be convenient when some program has left the scroll regions set incorrectly (often a problem when using VMS or TOPS-20). The full reset entry will clear the screen, reset tabs to every eight columns, and reset the terminal modes (such as wrap and smooth scroll) to their initial states just after **xterm** has finished processing the command line options. The **Tektronix** menu sets various modes in the Tektronix emulation, and is popped up when the control key and pointer button two are pressed in the Tektronix window. The current font size is checked in the modes section of the menu. The **PAGE** entry in the command section clears the Tektronix window.

OTHER FEATURES

Xterm automatically highlights the window border and text cursor when the pointer enters the window (selected) and unhighlights them when the pointer leaves the window (unselected). If the window is the focus window, then the window is highlighted no matter where the pointer is.

In VT102 mode, there are escape sequences to activate and deactivate an alternate screen buffer, which is the same size as the display area of the window. When activated, the current screen is saved and replaced with the alternate screen. Saving of lines scrolled off the top of the window is disabled until the normal screen is restored. The **termcap**(5) entry for **xterm** allows the visual editor **vi**(1) to switch to the alternate

screen for editing, and restores 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 bitmap 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), **termcap**(5), **vi**(1), **curses**(3X), **more**(1).

BUGS

Xterm will hang forever if you try to paste too much text at one time. It is both producer and consumer for the **pty** and can deadlock.

Variable-width fonts are not handled reasonably.

This program still needs to be rewritten. It should be split into very modular sections, with the various emulators being completely separate widgets that don't know about each other. Ideally, you'd like to be able to pick and choose emulator widgets and stick them into a single control widget.

The focus is considered lost if some other client (e.g., the window manager) grabs the pointer; it is difficult to do better without an addition to the protocol.

There needs to be a dialog box to allow entry of log file name and the COPY file name.

Many of the options are not resettable after **xterm** starts.

This manual page is too long. There should be a separate users manual defining all of the non-standard escape sequences.

All programs should be written to use X directly; then we could eliminate this program.

NAME

xwd – dump an image of an X window

SYNTAX

xwd [-help] [-nobdrs] [-out *file*] [-xy] [-display *display*]

DESCRIPTION

Xwd is an X Window System window dumping utility. **Xwd** allows X users to store window images in a specially formated 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

–help	Prints out the "Usage:" command syntax summary.
–nobdrs	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*	Allows you to explicitly specify the output file on the command line. The default is to output to standard out.
–xy	Applies to color displays only. It selects "XY" format dumping instead of the default "Z" format.
–display *display*	Allows you to specify the server to connect to (see X(1)).

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), **xdpr**(1), **X**(1)

NAME

xwininfo – window information utility for X

SYNTAX

xwininfo [-help] [-id *id*] [-root] [-name *name*] [-int] [-tree] [-stats] [-bits] [-events] [-size] [-wm] [-all] [-display *display*]

DESCRIPTION

Xwininfo is a utility for displaying information about windows. Depending on which options are chosen, various information is displayed. 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. In addition, if it is easier, 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 Prints out the "Usage:" command syntax summary.

–id *id* 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* 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 Specifies that X's root window is the target window. This is useful in situations where the root window is completely obscured.

–int Specifies that all X window ids should be displayed as integer values. The default is to display them as hexadecimal values.

–tree Causes the selected window's root, parent, and children windows id's and name's to be displayed.

–stats Causes various attributes having to do with location and appearance of the selected window to be displayed. Information displayed includes the location of the window, its width and height, its depth, border width, class, and map state.

–bits Causes various attributes of the selected window having to do with raw bits and how it is to be stored to be displayed. Information displayed includes the window's window and bit gravities, the window's backing store hint and *backing_planes* value, its backing pixel, and whether or not the window has save-under set.

–events 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 prograte are displayed.

–size Causes the selected window's sizing hints to be displayed. Information displayed for both the normal size hints and the zoom size hints includes: 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 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.

–all Asks for all information possible.

–display *display* 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 you wish
     ==> information on by clicking the
     ==> mouse in that window.

xwininfo ==> Window id: 0x8006b (fred)

     ==> Upper left X: 0
     ==> Upper left Y: 0
     ==> Width: 1024
     ==> Height: 864
     ==> Depth: 1
     ==> Border width: 0
     ==> Window class: InputOutput
     ==> Window Map State: IsUnviewable
```

ENVIRONMENT

DISPLAY To get default host and display number.

SEE ALSO

X(1), **xprop**(1)

NAME

xwud – image displayer for X

SYNTAX

xwud [-help] [-inverse] [-in *file*] [-display *display*]

DESCRIPTION

Xwud is an X Window System window image undumping utility. **Xwud** allows X users to display window images that were saved in a specially formatted dump file. The window image will appear at the coordinates of the original window from which the dump was taken. This is a crude version of a more advanced utility that has never been written. Monochrome dump files are displayed on a color monitor in the default foreground and background colors.

OPTIONS

–help	Prints out a short description of the allowable options.
–in *file*	Allows you to explicitly specify the input file on the command line. The default is to take input from standard in.
–inverse	Applies to monochrome window dump files only. If selected, the window is undumped in reverse video. This is mainly needed because the display is "write white," whereas dump files intended eventually to be written to a printer are generally "write black."
–display *display*	Allows you to specify the server to connect to (see **X**(1)).

ENVIRONMENT

DISPLAY	To get default display.

FILES

XWDFile.h	X Window Dump File format definition file.

BUGS

Does not attempt to do color translation when the destination screen does not have a colormap exactly matching that of the original window.

SEE ALSO

xwd(1), **xpr**(1), **xdpr**(1), **X**(1)

TABLE OF CONTENTS

3X. Xlib Routines

NAME

XAddHost, XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XSetAccessControl, XEnableAccessControl, XDisableAccessControl – control host access

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*;

OPTIONS

display	Specifies the connection to the X server.
mode	Specifies whether you want to change the access control to enable or disable. **EnableAccess** enables host access control. **DisableAccess** disables host access control.
host	Specifies the network address of the host machine.
hosts	Specifies each host that is to be added.
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 (enabled or disabled).

DESCRIPTION

The **XAddHost** function adds the specified host to the access control list for that display. The display hardware must be on the same host as the program issuing the command.

The **XAddHosts** function adds each specified host to the access control list for that display. The display hardware must be on the same host as the program issuing the command.

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.

The **XRemoveHost** function removes the specified host from the access control list for that display. The display hardware must be on the same host as the client process.

The **XRemoveHosts** function removes each specified host from the access control list for that display. The display hardware must be on the same host as the client process.

The **XSetAccessControl** function either enables or disables the use of the access control list at connection setups.

The **XEnableAccessControl** function enables the use of the access control list at connection setups.

The **XDisableAccessControl** function disables the use of the access control list at connection setups.

XAddHost, XAddHosts, XRemoveHost, and **XRemoveHosts** can generate **BadAlloc** and **BadValue** errors.

XSetAccessControl can generate **BadAlloc** and **BadValue** errors.

XEnableAccessControl and **XDisableAccessControl** can generate a **BadAccess** error.

DIAGNOSTICS

BadAccess A client attempted to modify the access control list from other than the local (or otherwise authorized) host.

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

Xlib – C Language X Interface

NAME

XAllocColor, XAllocNamedColor, XAllocColorCells, XAllocColorPlanes, XFreeColors – allocate and free colors

SYNTAX

Status **XAllocColor**(*display*, *cmap*, *screen_in_out*)
Display **display*;
Colormap *cmap*;
XColor **screen_in_out*;

Status **XAllocNamedColor**(*display*, *cmap*, *color_name*, *visual_def_return*, *exact_def_return*)
Display **display*;
Colormap *cmap*;
char **color_name*;
XColor **visual_def_return*, **exact_def_return*;

Status **XAllocColorCells**(*display*, *cmap*, *contig*, *plane_masks_return*, *nplanes*, *pixels_return*, *ncolors*)
Display **display*;
Colormap *cmap*;
Bool *contig*;
unsigned long *plane_masks_return*[];
unsigned int *nplanes*;
unsigned long *pixels_return*[];
unsigned int *ncolors*;

Status **XAllocColorPlanes**(*display*, *cmap*, *contig*, *pixels_return*, *ncolors*, *nreds*, *ngreens*, *nblues*, *rmask_return*, *gmask_return*, *bmask_return*)
Display **display*;
Colormap *cmap*;
Bool *contig*;
unsigned long *pixels_return*[];
int *ncolors*;
int *nreds*, *ngreens*, *nblues*;
unsigned long **rmask_return*, **gmask_return*, **bmask_return*;

XFreeColors(*display*, *cmap*, *pixels*, *npixels*, *planes*)
Display **display*;
Colormap *cmap*;
unsigned long *pixels[]*;
int *npixels*;
unsigned long *planes*;

OPTIONS

cmap	Specifies the colormap ID.
color_name	Specifies the color name string (for example, "red") whose color definition structure you want returned.
contig	Specifies a boolean value. You pass the value 1 if the planes must be contiguous or the value 0 if the planes do not need to 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**	Specifies the number of red, green, and blue colors (shades). The value you pass must be non-negative.
pixels	Specifies an array of pixel values.
pixels_return	Returns an array of pixel values.
plane_masks_return	Returns an array of plane masks.
planes	Specifies the planes you want to free.
rmask_return **gmask_return** **bmask_return**	Returns bit masks for the red, green, and blue planes.
screen_in_out	Specifies or returns the values actually used in the colormap.
visual_def_return	Returns the closest RGB values provided by the hardware.

DESCRIPTION

The **XAllocColor** function allocates a read-only colormap entry corresponding to the closest red, green, and blue 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 red, green, and blue values actually used. **XAllocColor** can generate **BadAlloc** and **BadColor** errors.

The **XAllocNamedColor** function looks up the named color with respect to the screen that is associated with the specified colormap. Both the exact data base definition and the closest color supported by the screen are returned. **XAllocNamedColor** can generate **BadAlloc** and **BadColor** errors.

The **XAllocColorCells** function allocates read/write color cells. The number of colors must be positive and the number of planes nonnegative. **XAllocColorCells** can generate **BadAlloc, BadColor,** and **BadValue** errors.

The **XAllocColorPlanes** function allocates read/write color resources in a way that is compatible with **DirectColor** displays. **XAllocColorPlanes** can generate **BadAlloc, BadColor,** and **BadValue** errors.

The **XFreeColors** function frees the cells represented by pixels whose values are in the pixels array. **XFreeColors** can generate **BadAccess, BadColor,** and **BadValue** errors.

DIAGNOSTICS

BadAccess	A client attempted to free a colormap entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only colormap entry.
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.
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

XCreateColormap(3X), **XQueryColor**(3X), **XStoreColors**(3X)
Xlib – C Language X Interface

NAME

XAllowEvents – continue frozen event processing

SYNTAX

XAllowEvents(*display*, *event_mode*, *time*)
Display **display*;
int *event_mode*;
Time *time*;

OPTIONS

display	Specifies the connection to the X server.
event_mode	Specifies the event mode. You can pass one of these constants: **AsyncPointer, SyncPointer, AsyncKeyboard, SyncKeyboard, ReplayPointer, ReplayKeyboard, AsyncBoth,** or **SyncBoth.**
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or **CurrentTime.**

DESCRIPTION

The **XAllowEvents** function releases some queued events if the client has caused a device to freeze. **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

NAME

XChangeKeyboardControl, XGetKeyboardControl, XAutoRepeatOn, XAutoRepeatOff, XBell, XQueryKeymap – manipulate keyboard settings

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];

OPTIONS

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 base volume for the bell, which can range from -100 to 100 inclusive.
value_mask	Specifies one value for each one bit in the mask (least to most significant bit). The values are associated with the set of keys for the previously specified keyboard.
values	Specifies a pointer to the structure **XKeyboardControl.**
values_return	Returns the current keyboard parameter in the specified **XKeyboardState** structure.

DESCRIPTION

The **XChangeKeyboardControl** function controls the keyboard characteristics defined by the **XKeyboardControl** structure. **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. **XBell** can generate a **BadValue** error.

The **XQueryKeymap** function returns a bit vector for the logical state of the keyboard, where each one bit indicates that the corresponding key is currently pressed down.

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(3X), **XSetPointerMapping**(3X)
Xlib – C Language X Interface

NAME

XChangeKeyboardMapping, XGetKeyboardMapping, XSetModifierMapping, XGetModifierMapping, XNewModifierMap, XInsertModifiermapEntry, XDeleteModifiermapEntry, XFreeModifierMap – manipulate keyboard encoding

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_wanted, keycode_count, keysyms_per_keycode_return*)
Display **display*;
KeyCode *first_keycode_wanted*;
int *keycode_count*;
int **keysyms_per_keycode_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, keysym_entry, modifier*)
XModifierKeymap **modmap*;
KeyCode *keysym_entry*;
int *modifier*;

XModifierKeymap ***XDeleteModifiermapEntry**(*modmap, keysym_entry, modifier*)
XModifierKeymap **modmap*;
KeyCode *keysym_entry*;
int *modifier*;

XFreeModifierMap(*modmap*)
XModifierKeymap **modmap*;

OPTIONS

keycode_count	Specifies the number of keycodes that are to be returned.
display	Specifies the connection to the X server.
first_keycode	Specifies the first keycode that is to be changed.
first_keycode_wanted	Specifies the first keycode that is to be returned.
keysyms	Specifies a pointer to an array of keysyms.
keysym_entry	Specifies the keysyms.
keysyms_per_keycode	Specifies the keysyms that are to be used.
keysyms_per_keycode_return	Returns the number of keysyms per keycode.
max_keys_per_mod	Specifies the maximum number of keycodes assigned to any of the modifiers in the map.

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, starting with **first_keycode,** defines the symbols for the specified number of keycodes. **XChangeKeyboardMapping** can generate **BadAlloc, BadLength,** and **BadValue** errors.

The **XGetKeyboardMapping** function, starting with **first_keycode,** returns the symbols for the specified number of keycodes. **XGetKeyboardMapping** can generate a **BadValue** error.

The **XSetModifierMapping** function specifies the keycodes of the keys, if any, that are to be used as modifiers. **XSetModifierMapping** can generate **BadAlloc** and **BadValue** errors.

The **XGetModifierMapping** function returns a newly created **XModifierKeymap** structure that contains the keys being used as modifiers.

The **XNewModifierMapping** function returns a **XModifierKeymap** structure.

The **XInsertModifiermapEntry** function add 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 the resulting **XModifierKeymap** structure.

The **XFreeModifierMapping** function frees the specified **XModifierKeymap** structure.

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

XSetPointerMapping(3X)
Xlib – C Language X Interface

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*;

OPTIONS

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 set. You can pass one of these constants: **True** or **False.**
do_threshold	Specifies a boolean value that controls whether the value for the **accel_numerator** or **accel_denominator** are set. You can pass one of these constants: **True** or **False.**
threshold	Specifies the acceleration threshold.
threshold_return	Returns the acceleration threshold.

DESCRIPTION

The **XChangePointerControl** function defines how the pointing device moves. **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

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_add*)
Display **display*;
Window *w_add*;

XRemoveFromSaveSet(*display, w_remove*)
Display **display*;
Window *w_remove*;

OPTIONS

change_mode	Specifies the mode. You can pass one of **SetModeInsert** or **SetModeDelete.** If **SetModeInsert, XChangeSaveSet** adds the window to this client's save-set. If **SetModeDelete, XChangeSaveSet** deletes the window from this client's save-set.
display	Specifies the connection to the X server.
w	Specifies the window ID.
w_add	Specifies the window ID of the window whose children you want to add to the client's save-set.
w_remove	Specifies the window ID of the window whose children you want to remove from the client's save-set.

DESCRIPTION

Depending on the constant you passed to the **change_mode** argument, the **XChangeSaveSet** function either adds or removes a subwindow from the client's save-set. **XChangeSaveSet** can generate **BadMatch, BadValue,** and **BadWindow** errors.

The **XAddToSaveSet** function adds the children of the specified window to the client's save-set. **XAddToSaveSet** can generate **BadMatch** and **BadWindow** errors.

The **XRemoveFromSaveSet** function removes the children of the specified window from the client's save-set. **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(3X)
Xlib – C Language X Interface

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*;

OPTIONS

attributes	Attributes of the window to be set at creation time should be set in this structure. The valuemask should have the appropriate bits set to indicate which attributes have been set in the structure.
background_pixel	Specifies the pixel of the background. This pixel value determines which entry in the colormap is used.
background_pixmap	Specifies the background pixmap. If a Pixmap ID is specified, the background is painted with this pixmap. If **None,** no background is painted. If **ParentRelative,** the parent's pixmap is used.
border_pixel	Specifies the entry in the colormap.
border_pixmap	Specifies the border pixmap. If you specify a pixmap ID, the associated pixmap is used for the border. If **CopyFromParent** is specified, a copy of the parent window's border pixmap is used.
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 ID.

DESCRIPTION

Depending on the valuemask, the **XChangeWindowAttributes** function uses the window attributes in the **XSetWindowAttributes** structure to change the specified window attributes. **XChangeWindowAttributes** can generate **BadAccess, BadColor, BadCursor, BadMatch, BadPixmap, BadValue,** and

BadWindow errors.

The **XSetWindowBackground** function sets the background pixel of the window to the pixel value you specify. **XSetWindowBackground** can generate **BadMatch** and **BadWindow** errors.

The **XSetWindowBackgroundPixmap** function sets the background pixmap of the window to the pixmap you specify. **XSetWindowBackgroundPixmap** can generate **BadColor, BadMatch, BadPixmap,** and **BadWindow** errors.

The **XSetWindowBorder** function sets the border pixel of the window to the pixel value you specify. It uses this value as an entry into the colormap to determine which color is to be used to paint the border. **XSetWindowBorder** can generate **BadMatch, BadPixmap, BadValue,** and **BadWindow** errors.

The **XSetWindowBorderPixmap** function sets the border pixmap of the window to the pixmap you specify. It uses this entry for the border. **XSetWindowBorderPixmap** can generate **BadMatch, BadPixmap, BadValue,** and **BadWindow** errors.

DIAGNOSTICS

BadAccess	A client attempted to free a colormap entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only colormap 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(3X), **XCreateWindow**(3X), **XDestroyWindow**(3X), **XMapWindow**(3X), **XRaiseWindow**(3X), **XUnmapWindow**(3X)
Xlib – C Language X Interface

NAME

XClearArea, XClearWindow – clear windows

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*;

OPTIONS

display	Specifies the connection to the X server.
exposures	Specifies a boolean value of **True** or **False.**
w	Specifies the window ID.
width **height**	Specify the width and height.
x **y**	Specify the x and y coordinates.

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. **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**). **XClearWindow** can generate **BadMatch, BadValue,** 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(3X)
Xlib – C Language X Interface

NAME

XConfigureWindow, XMoveWindow, XResizeWindow, XMoveResizeWindow, XSetWindowBorderWidth – configure windows

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*;

OPTIONS

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 change window values bits.
values	Specifies a pointer to the structure **XWindowChanges.**
w	Specifies the window ID.
width **height**	Specifies the width and height.
x **y**	Specifies the x and y coordinates.

DESCRIPTION

The **XConfigureWindow** function uses the values specified in the **XWindowChanges** structure to reconfigure a window's size, position, border, and stacking order. **XConfigureWindow** can generate **BadMatch, BadValue,** and **BadWindow** errors.

The **XMoveWindow** function moves the specified window to the specified x and y coordinates. This function does not change the window's size, does not raise the window, and does not change the mapping state of the window. **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 raise the window. **XResizeWindow** can generate a **BadWindow** error.

The **XMoveResizeWindow** function changes the size and location of the specified window without raising it. **XMoveResizeWindow** can generate **BadMatch, BadValue,** and **BadWindow** errors.

The **XSetWindowBorderWidth** function sets the specified window's border width to the specified width. **XSetWindowBorderWidth** can generate **BadValue** and **BadWindow** errors.

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(3X), **XCreateWindow**(3X), **XDestroyWindow**(3X), **XMapWindow**(3X), **XRaiseWindow**(3X), **XUnmapWindow**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

dest_x
dest_y Specifies the x and y coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the destination rectangle.

display Specifies the connection to the X server.

gc Specifies the graphics context.

plane Specifies the bit-plane. You must set exactly one bit.

src
dest Specifies the source and destination rectangles to be combined.

src_x
src_y Specifies the x and y coordinates of the source rectangle relative to its origin. These coordinates specify the upper-left corner of the source rectangle.

width
height Specifies the width and height.

DESCRIPTION

The **XCopyArea** function combines the specified rectangle of **src** with the specified rectangle of **dest.** **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. **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(3X)
Xlib – C Language X Interface

NAME

XCreateColormap, XCopyColormapAndFree, XFreeColormap, XSetWindowColormap – create, copy, or destroy colormaps

SYNTAX

Colormap **XCreateColormap** (*display*, *w*, *visual*, *alloc*)
Display **display*;
Window *w*;
Visual **visual*;
int *alloc*;

Colormap **XCopyColormapAndFree** (*display*, *cmap*)
Display **display*;
Colormap *cmap*;

XFreeColormap (*display*, *cmap*)
Display **display*;
Colormap *cmap*;

XSetWindowColormap (*display*, *w*, *cmap*)
Display **display*;
Window *w*;
Colormap *cmap*;

OPTIONS

alloc	Specifies the colormap entries to be allocated. You can pass one of these constants: **AllocNone** or **AllocAll.**
cmap	Specifies the colormap ID.
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, the function returns a **BadMatch** error.
w	Specifies the window ID.

DESCRIPTION

The **XCreateColormap** function creates a colormap of the specified visual type for the screen on which the specified window resides and associates the colormap ID with it. **XCreateColormap** can generate **BadAlloc, BadMatch, BadValue,** and **BadWindow** errors.

The **XCopyColormapAndFree** function obtains a new colormap when allocating out of a previous colormap has failed due to resource exhaustion (that is, too many cells or planes were in use in the original colormap). **XCopyColormapAndFree** can generate **BadAlloc** and **BadColor** errors.

The **XFreeColormap** function deletes the association between the colormap resource ID and the colormap. However, this function has no effect on the default colormap for a screen. **XFreeColormap** can generate a **BadColor** error.

The **XSetWindowColormap** function sets the specified colormap of the specified window. **XSetWindowColormap** can generate **BadColor, BadMatch,** and **BadWindow** errors.

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(3X), XQueryColor(3X), XStoreColors(3X)
Xlib – C Language X Interface

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;
```

OPTIONS

background_color	Specifies the red, green, and blue (RGB) values for the background of the source.
display	Specifies the connection to the X server.
mask	Specifies the source bits of the cursor that are to be displayed. You can also pass **None.**
mask_char	Specifies the glyph character for the mask.
mask_font	Specifies the font for the mask glyph. You can also pass **None.**
source_char	Specifies the character glyph for the source.
foreground_color	Specifies the red, green, and blue (RGB) values for the foreground of the source.
source_font	Specifies the font for the source glyph.
shape	Specifies the shape in which you want to create the standard cursor.
source	Specifies the shape of the source cursor.
x **y**	Specifies the x and y coordinates.

DESCRIPTION

The **XCreateFontCursor** function creates a cursor from a standard font. **XCreateFontCursor** can generate **BadAlloc, BadMatch,** and **BadValue** errors.

The **XCreatePixmapCursor** function creates a cursor and returns the cursor ID associated with it. **XCreatePixmapCursor** can generate **BadAlloc, BadMatch,** and **BadPixmap** errors.

The **XCreateGlyphCursor** function is similar to **XCreatePixmapCursor** and creates a cursor from font glyphs. For **XCreateGlyphCursor,** however, the source and mask bitmaps are obtained from the specified font glyphs. **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(3X), **XRecolorCursor**(3X)
Xlib – C Language X Interface

NAME

XCreateGC, XCopyGC, XChangeGC, XFreeGC – create and free graphics contexts

SYNTAX

GC **XCreateGC**(*display, d, valuemask_create, values*)
Display **display*;
Drawable *d*;
unsigned long *valuemask_create*;
XGCValues **values*;

XCopyGC(*display, src, valuemask_copy, dest*)
Display **display*;
GC *src, dest*;
unsigned long *valuemask_copy*;

XChangeGC(*display, gc, valuemask_change, values*)
Display **display*;
GC *gc*;
unsigned long *valuemask_change*;
XGCValues **values*;

XFreeGC(*display, gc*)
Display **display*;
GC *gc*;

OPTIONS

d	Specifies the drawable.
dest	Specifies the destination graphics context.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
src	Specifies the components of the source graphics context.
valuemask_change	Specifies which components in the graphics context are to be changed using information in the **XGCValues** structure. This argument is the bitwise inclusive OR of one or more of the valid GC component masks.
valuemask_copy	Specifies which components in the source graphics context are to be copied to the destination graphics context. This argument is the bitwise inclusive OR of one or more of the valid GC component masks.
valuemask_create	Specifies which components in the graphics context are to be set using information in the **XGCValues** structure. This argument is the bitwise inclusive OR of one or more of the valid GC component masks.
values	Specifies a pointer to the **XGCValues** structure.

DESCRIPTION

The **XCreateGC** function creates a graphics context and returns a GC. **XCreateGC** can generate **BadAlloc, BadDrawable, BadFont, BadMatch, BadPixmap,** and **BadValue** errors.

The **XCopyGC** function copies the specified components from the source graphics context to the destination graphics context. **XCopyGC** can generate **BadAlloc, BadGC, BadMatch,** and **BadValue** errors.

The **XChangeGC** function changes the components specified by the **valuemask_change** argument for the specified graphics context. **XChangeGC** can generate **BadAlloc, BadFont, BadGC, BadMatch, BadPixmap,** and **BadValue** errors.

The **XFreeGC** function destroys the specified graphics context as well as the shadow copy.

XFreeGC can generate a **BadGC** 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.
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

XQueryBestSize(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetState**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

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*;

int __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*;

int __XAddPixel__(*ximage, value*)
XImage **ximage*;
int *value*;

int __XDestroyImage__(*ximage*)
XImage **ximage*;

OPTIONS

bytes_per_line Specifies the number of bytes in the client image between the start of one scanline and the start of the next. If you pass a zero value, Xlib assumes that the scanlines are contiguous in memory and calculates the value of **bytes_per_line** itself.

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 one of these constants: **XYPixmap** or **ZPixmap.**

height Specifies the height (in pixels) of the image.

offset Specifies the number of pixels to ignore at the beginning of the scanline. This permits the rapid displaying of the image without requiring each scanline to be

shifted into position.

pixel Specifies the new pixel value.

subimage_height Specifies the height (in pixels) of the new subimage.

subimage_width Specifies the width (in pixels) of the new subimage.

value Specifies the constant value that is to be added.

visual Specifies a pointer to the visual.

width Specifies the width (in pixels) of the image.

ximage Specifies a pointer to the image.

bitmap_pad Specifies the quantum of a scanline. 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. You must pass one of these values: 8, 16, or 32.

x
y Specifies the x and y coordinates.

DESCRIPTION

The **XCreateImage** function allocates the memory needed for an **XImage** structure for the specified display.

The **XGetPixel** function returns the specified pixel from the named image.

The **XPutPixel** function overwrites the pixel in the named image with the specified pixel value.

The **XSubImage** function creates a new image that is a subsection of an existing one.

The **XAddPixel** function adds a constant value to every pixel in an image.

The **XDestroyImage** function deallocates the memory associated with the **XImage** structure.

SEE ALSO

XPutImage(3X)
Xlib – C Language X Interface

NAME

XCreatePixmap, XFreePixmap – create and 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*;

OPTIONS

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**	Specifies the width and height.

DESCRIPTION

The **XCreatePixmap** function creates a pixmap of the width, height, and depth you specified. It also assigns the pixmap ID to it. It is valid to pass a window whose class is **InputOnly** to the drawable argument. **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 no other resources reference 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

Xlib – C Language X Interface

NAME

XCreateRegion, XSetRegion, XDestroyRegion – create and destroy regions

SYNTAX

Region **XCreateRegion**()

XSetRegion(*display*, *gc*, *r*)
Display **display*;
GC *gc*;
Region *r*;

XDestroyRegion(*r*)
Region *r*;

OPTIONS

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
r	Specifies the region.

DESCRIPTION

The **XCreateRegion** function creates a new empty region.

The **XSetRegion** function sets the clip mask in the graphics contexts 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(3X), **XIntersectRegion**(3X)
Xlib – C Language X Interface

NAME

XCreateWindow, XCreateSimpleWindow – create windows

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*, *border_width*;
unsigned long *border*;
unsigned long *background*;

OPTIONS

attributes Attributes of the window to be set at creation time should be set in this structure. The **valuemask** 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, in pixels, the width of the created window's border. The **border_width** for an **InputOnly** window must be zero. Otherwise, a **BadMatch** error is returned.

class Specifies the created window's class. You can pass one of these constants: **InputOutput, InputOnly,** or **CopyFromParent.** A class of **CopyFromParent** means the class is taken from the parent.

depth A depth of zero for class **InputOutput** or **CopyFromParent** means the depth is taken from the parent.

display Specifies the connection to the X server.

parent Specifies the parent window ID.

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 Specifies the width and height.

x

y Specifies the x and y coordinates.

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. **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. **XCreateSimpleWindow** can generate **BadAlloc, BadMatch, BadValue,** and **BadWindow** errors.

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(3X), **XConfigureWindow**(3X), **XDestroyWindow**(3X), **XMapWindow**(3X), **XRaiseWindow**(3X), **XUnmapWindow**(3X)
Xlib – C Language X Interface

NAME

XDefineCursor, XUndefineCursor – define cursors

SYNTAX

XDefineCursor(*display, w, cursor*)
Display **display*;
Window *w*;
Cursor *cursor*;

XUndefineCursor(*display, w*)
Display **display*;
Window *w*;

OPTIONS

cursor	Specifies the cursor that is to be displayed when the pointer is in the specified window. You can pass **None** if no cursor is to be displayed.
display	Specifies the connection to the X server.
w	Specifies the window ID.

DESCRIPTION

The **XDefineCursor** function defines the mouse cursor. **XDefineCursor** can generate **BadAlloc, BadCursor,** and **BadWindow** errors.

The **XUndefineCursor** undoes the effect of a previous **XDefineCursor** for this window. **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(3X), **XRecolorCursor**(3X)
Xlib – C Language X Interface

NAME

XDestroyWindow, XDestroySubwindows – destroy windows

SYNTAX

XDestroyWindow(*display*, *w*)
Display **display*;
Window *w*;

XDestroySubwindows(*display*, *w*)
Display **display*;
Window *w*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.

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. **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 sever to generate exposure events on the specified window. **XDestroySubwindows** can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a window argument does not name a defined window.

SEE ALSO

XChangeWindowAttributes(3X), **XConfigureWindow**(3X), **XCreateWindow**(3X), **XMapWindow**(3X), **XRaiseWindow**(3X), **XUnmapWindow**(3X)
Xlib – C Language X Interface

NAME

XDrawArc, XDrawArcs – draw arcs

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*;

OPTIONS

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 graphics context.
narcs	Specifies the number of arcs in the array.
width **height**	Specifies the width and height.
x **y**	Specifies the x and y coordinates.

DESCRIPTION

XDrawArc draws a single circular or elliptical arc, while **XDrawArcs** draws multiple circular or elliptical arcs. **XDrawArc** and **XDrawArcs** 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

XDrawLine(3X), **XDrawPoint**(3X), **XDrawRectangle**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
length	Specifies the number of characters in the string argument.
string	Specifies the character string.
x **y**	Specifies the x and y coordinates.

DESCRIPTION

The **XDrawImageString** and **XDrawImageString16** functions draw 8-bit and 16-bit image text characters in the specified drawable. These functions also modify both the foreground and background pixels in the characters. **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(3X), **XDrawText**(3X)
Xlib – C Language X Interface

NAME

XDrawLine, XDrawLines, XDrawSegments – draw lines and polygons

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*;

OPTIONS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
mode	Specifies the coordinate mode. **CoordModeOrigin** treats a coordinate as related to the origin, while **CoordModePrevious** treats all coordinates after the first as relative to the previous point.
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**	Specifies the points used to connect the line. Thus, **XDrawLine** draws a line connecting point x1, y1 to point x2, y2.

DESCRIPTION

The **XDrawLine** function draws a single line between two points in the specified drawable.

The **XDrawLines** function draws multiple lines in the specified drawable.

The **XDrawSegments** function draws multiple, but not necessarily connected, lines in the specified drawable.

XDrawLine, XDrawLines, and **XDrawSegments** can generate **BadDrawable, BadGC,** and **BadMatch** errors.

XDrawLines can also generate a **BadValue** error.

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(3X), **XDrawPoint**(3X), **XDrawRectangle**(3X)
Xlib – C Language X Interface

NAME

XDrawPoint, XDrawPoints – draw points

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*;

OPTIONS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
mode	Specifies the coordinate mode. **CoordModeOrigin** treats a coordinate as related to the origin, while **CoordModePrevious** treats all coordinates after the first as relative to the previous point.
npoints	Specifies the number of points in the array.
points	Specifies a pointer to an array of points.
x **y**	Specifies the x and y coordinates where you want the point drawn.

DESCRIPTION

The **XDrawPoint** function uses the foreground pixel and function components of the graphics context to draw a single point into the specified drawable, while **XDrawPoints** draws multiple points into the specified drawable. These functions are not affected by the tile or stipple in the graphics context.

XDrawPoint can generate **BadDrawable, BadGC,** and **BadMatch** errors. **XDrawPoint** 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

XDrawArc(3X), **XDrawLine**(3X), **XDrawRectangle**(3X)
Xlib – C Language X Interface

NAME

XDrawRectangle, XDrawRectangles – draw rectangles

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*;

OPTIONS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
nrectangles	Specifies the number of rectangles in the array.
rectangles	Specifies a pointer to an array of rectangles.
width **height**	Specifies the width and height that define the outline of the rectangle.
x **y**	Specifies the x and y coordinates that define 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 were specified for each rectangle:

[x,y,] [x+width,y] [x+width,y+height] [x,y+height] [x,y] **XDrawRectangle** and **XDrawRectangles** 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

XDrawArc(3X), **XDrawLine**(3X), **XDrawPoint**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
length	Specifies the number of characters in the string argument.
string	Specifies the character string.
x **y**	Specifies the x and y coordinates.

DESCRIPTION

The **XDrawString** and **XDrawString16** functions draw 8-bit and 16-bit text characters in the specified drawable. **XDrawString** and **XDrawString16** can generate **BadDrawable, BadFont, 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(3X), **XDrawText**(3X)
Xlib – C Language X Interface

NAME

XDrawText, XDrawText16 – draw polytext text

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*;

OPTIONS

d	Specifies the drawable.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
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.

DESCRIPTION

The **XDrawText** and **XDrawText16** functions draw 8-bit and 16-bit polytext characters in the specified drawable. **XDrawText** and **XDrawText16** can generate **BadDrawable, BadFont, BadGC,** and **BadMatch** errors.

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(3X), XDrawString(3X)
Xlib – C Language X Interface

NAME

XEmptyRegion, XEqualRegion – determine if regions are empty or equal

SYNTAX

int XEmptyRegion(*r*)
Region *r*;

int XEqualRegion(*r1*, *r2*)
Region *r1*, *r2*;

OPTIONS

r	Specifies the region.
r1 **r2**	Specify the two regions.

DESCRIPTION

The **XEmptyRegion** function returns nonzero if the specified region is empty.

The **XEqualRegion** function returns nonzero if two regions have the same offset, size, and shape.

SEE ALSO

XCreateRegion(3X), XIntersectRegion(3X)
Xlib – C Language X Interface

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*;

OPTIONS

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 graphics context.
mode	Specifies the coordinate mode. **CoordModeOrigin** treats a coordinate as related to the origin, while **CoordModePrevious** treats all coordinates after the first as relative to the previous point.

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 an argument that helps the server to improve performance. You can pass one of these constants: **Complex, Convex,** or **Nonconvex.**

width
height Specifies the width and height.

x
y Specifies the x and y coordinates.

DESCRIPTION

The **XFillRectangle** and **XFillRectangles** functions fill the specified rectangle or rectangles as if a four-point **XFillPolygon** were specified for each rectangle:

[x,y] [x+width,y] [x+width,y+height] [x,y+height]

XFillRectangle and **XFillRectangles** can generate **BadDrawable, BadGC,** and **BadMatch** errors.

The **XFillPolygon** function fills a polygon area in the specified drawable. **XFillRectangle** can generate **BadDrawable, BadGC, BadMatch,** and **BadValue** errors.

The **XFillArc** function fills a single arc in the specified drawable, while **XFillArcs** fills multiple arcs in the specified drawable. **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(3X), **XDrawRectangle**(3X)
Xlib – C Language X Interface

NAME

XFlush, XSync, XEventsQueued, XPending, XNextEvent, XPeekEvent – basic error handling

SYNTAX

XFlush(*display*)
Display **display*;

XSync(*display, discard*)
Display **display*;
int *discard*;

int **XEventQueued**(*display, more*)
Display **display*;
int *mode*;

int **XPending**(*display*)
Display **display*;

XNextEvent(*display, event_return*)
Display **display*;
XEvent **event_return*;

XPeekEvent(*display, event_return*)
Display **display*;
XEvent **event_return*;

OPTIONS

discard	Specifies whether **XSync** discards all events on the event queue. You can pass the value 0 or 1.
display	Specifies the connection to the X server.
event_return	Copies the event's associated structure into this client-supplied structure.
mode	Specifies the mode. You can specify one of these constants: **QueuedAlready, QueuedAfterFlush, QueuedAfterReading.**

DESCRIPTION

The **XFlush** function flushes the output buffer.

The **XSync** function flushes the output buffer and then waits until all requests have been received and processed by the X server.

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,** it returns the number of events already in the queue, if it is nonzero. If there are no events in the queue, it flushes the output buffer, attempts to read more events out of the application's connection, and returns the number read. If mode is **QueuedAfterReading,** it returns the number of events already in the queue, if it is nonzero. If there are no events in the queue, it attempts to read more events out of the application's connection without flushing the output buffer and returns the number read.

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.

The **XNextEvent** function copies the first event from the event queue into the specified **XEvent** structure and then removes it from the queue.

The **XPeekEvent** function returns the first event from the event queue, but it does not remove the event from the queue.

SEE ALSO

XIfEvent(3X), **XPutBackEvent**(3X)
Xlib – C Language X Interface

NAME

XFree, XNoOp – free client data

SYNTAX

XFree(*data*)
char **data*;

XNoOp(*display*)
Display **display*;

OPTIONS

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.

The **XNoOp** function essentially sends a NoOperation request to the X server, thereby exercising the connection.

SEE ALSO

Xlib – C Language X Interface

NAME

XGetDefault – get X program defaults

SYNTAX

char ***XGetDefault**(*display*, *program*, *option*)
Display **display*;
char **program*;
char **option*;

OPTIONS

display	Specifies the connection to the X server.
option	Specifies the option name.
program	Specifies the program name for the Xlib defaults. You must pass the program name in with the program argument (usually argv[0]).

DESCRIPTION

The **XGetDefault** function finds out the fonts, colors, and other environment defaults favored by a particular user for the specified program.

SEE ALSO

XrmGetSearchList(3X)
Xlib – C Language X Interface

NAME

XGetVisualInfo, XMatchVisualInfo – obtain visual information

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*;

OPTIONS

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.
vinfo_mask	Specifies the visual mask value.
vinfo_return	Returns the match 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 the template argument. If no visual structures match the template using the specified **vinfo_mask, XGetVisualInfo** returns a NULL.

The **XMatchVisualInfo** function obtains the visual information that matches the specified depth and class of the screen.

SEE ALSO

Xlib – C Language X Interface

NAME

XGetWindowAttributes, XGetGeometry – get current window attribute or geometry

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*;

OPTIONS

border_width_return	Returns the border width in pixels.
d	Specifies the drawable.
depth_return	Returns the depth of the pixmap (bits per pixel for the object).
display	Specifies the connection to the X server.
root_return	Returns the root window ID for the specified window.
w	Specifies the window ID.
width_return **height_return**	Returns the drawable's dimensions (width and height).
window_attributes_return	Returns the specified window's attributes in the **XWindowAttributes** structure.
x_return **y_return**	Returns the x and y coordinates of the drawable. These coordinates 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.

The **XGetGeometry** function returns the root ID and the current geometry of the drawable. **XGetGeometry** can generate a **BadDrawable** error.

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(3X), **XQueryTree**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

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. This is 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 from the window. You can pass one of these constants: **True** or **False.**

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. 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.** Possible values are 8, 16, and 32.
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 data will be retrieved.
mode	Specifies the mode of the operation. You can pass one of these constants: **PropModeReplace, PropModePrepend,** or **PropModeAppend.**
nelements	Specifies the number of elements of the specified data format (8-bit, 16-bit, or 32-bit).
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 atom.
properties	Specifies the array of properties that are to be rotated.
req_type	Specifies the atom identifier associated with the property type. You can also pass **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 **XGetProperty.**
w	Specifies the window ID.

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** 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. **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** can generate **BadAlloc, BadAtom, BadMatch, BadValue,** and **BadWindow** errors.

The **XRotateWindowProperties** function allows you to rotate properties in the properties array and causes the X server to generate a **PropertyNotify** event. **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. **XDeleteProperty** 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

XInternAtom(3X)
Xlib – C Language X Interface

NAME

XGrabButton, XUngrabButton – manipulate the pointer

SYNTAX

XGrabButton(*display*, *button_grab*, *modifiers*, *grab_window*, *owner_events*, *event_mask*, *pointer_mode*, *keyboard_mode*, *confine_to*, *cursor*)

Display **display*;
unsigned int *button_grab*;
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_ungrab*, *modifiers*, *ungrab_window*)

Display **display*;
unsigned int *button_ungrab*;
unsigned int *modifiers*;
Window *ungrab_window*;

OPTIONS

button_grab	Specifies the pointer button that is to be grabbed when the specified modifier keys are down.
button_ungrab	Specifies the pointer button that is to be released in combination with the modifier keys.
confine_to	Specifies the window to confine the pointer in or **None** if it is not to be confined.
cursor	Specifies the cursor that is to be displayed during the grab.
display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. They can be the bitwise inclusive OR of these pointer event mask bits: **ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeyMapStateMask.**
grab_window	Specifies the window ID of the window you want to grab.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of these keymask bits: **ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask.**
owner_events	Specifies if the pointer events are to be reported normally (pass **True**) or with respect to the grab window if selected by the event mask (pass **False**).
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
ungrab_window	Specifies the window ID of the window you want to ungrab.

DESCRIPTION

The **XGrabButton** function establishes a passive grab. **XGrabButton** can generate **BadAlloc, 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. **XUngrabButton** can generate a **BadWindow** error.

DIAGNOSTICS

BadAccess	A client attempted to grab a key/button combination already grabbed by another client.
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(3X), XGrabPointer(3X), XGrabKey(3X), XGrabKeyboard(3X),
Xlib – C Language X Interface

NAME

XGrabKey, XUngrabKey – manipulate the keyboard

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*, *ungrab_window*)

Display **display*;
int *keycode*;
unsigned int *modifiers*;
Window *ungrab_window*;

OPTIONS

display	Specifies the connection to the X server.
grab_window	Specifies the window ID of the window associated with the keys you want to grab.
keycode	Specifies the keycode.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
modifiers	Specifies the set of keymasks. This mask is the bitwise inclusive OR of these keymask bits: **ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask.**
owner_events	Specifies a boolean value of either **True** or **False.**
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
ungrab_window	Specifies the window ID of the window associated with the keys you want to ungrab.

DESCRIPTION

The **XGrabKey** function establishes a passive grab on the keyboard. **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. **XUngrabKey** can generate a **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(3X), **XGrabButton**(3X), **XGrabKeyboard**(3X), **XGrabPointer**(3X)
Xlib – C Language X Interface

NAME

XGrabKeyboard, XUngrabKeyboard – manipulate 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*;

OPTIONS

display	Specifies the connection to the X server.
grab_window	Specifies the window ID of the window associated with the keyboard you want to grab.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
owner_events	Specifies a boolean value of either **True** or **False.**
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or **CurrentTime.**

DESCRIPTION

The **XGrabKeyboard** function actively grabs control of the main keyboard and generates **FocusIn** and **FocusOut** events. **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.**

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(3X), **XGrabButton**(3X), **XGrabKey**(3X), **XGrabPointer**(3X)
Xlib – C Language X Interface

NAME

XGrabPointer, XUngrabPointer, XChangeActivePointerGrab – manipulate 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*;

OPTIONS

confine_to	Specifies the window to confine the pointer in or **None** if it is not to be confined.
cursor	Specifies the cursor that is to be displayed during the grab.
display	Specifies the connection to the X server.
event_mask	Specifies which pointer events are reported to the client. They can be the bitwise inclusive OR of these pointer event mask bits: **ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeyMapStateMask.**
grab_window	Specifies the window ID of the window relative to which events are reported while it is grabbed.
keyboard_mode	Controls further processing of keyboard events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
owner_events	Specifies if the pointer events are to be reported normally (pass **True**) or with respect to the grab window if selected by the event mask (pass **False**).
pointer_mode	Controls further processing of pointer events. You can pass one of these constants: **GrabModeSync** or **GrabModeAsync.**
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or **CurrentTime.**

DESCRIPTION

The **XGrabPointer** function actively grabs control of the pointer and returns **GrabSuccess** if the grab was successful. **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.

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. **XChangeActivePointerGrab** can generate a **BadCursor** 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(3X), **XGrabButton**(3X), **XGrabKey**(3X), **XGrabKeyboard**(3X)
Xlib – C Language X Interface

NAME

XGrabServer, XUngrabServer – manipulate the keyboard

SYNTAX

XGrabServer(*display*)
Display **display*;

XUngrabServer(*display*)
Display **display*;

OPTIONS

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. It is recommended that you 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 server as much as possible.

SEE ALSO

XGrabButton(3X), **XGrabKey**(3X), **XGrabKeyboard**(3X), **XGrabPointer**(3X)
Xlib – C Language X Interface

NAME

XIfEvent, XCheckIfEvent, XPeekIfEvent – check the event queue

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*;

OPTIONS

arg	Specifies the user-supplied argument that is to be passed to the predicate procedure.
display	Specifies the connection to the X server.
event_return	Copies the matched event's associated structure into this client-supplied structure.
predicate	Specifies the procedure that is to be called to determine if the next event in the queue matches the one specified by the event argument.

DESCRIPTION

The **XIfEvent** function checks the event queue for the specified event. If the events match as determined by your predicate procedure, **XIfEvent** removes the event from the queue and, when it returns, copies the structure into the client-supplied **XEvent** structure.

The **XCheckIfEvent** function checks the event queue for the specified event.

The **XPeekIfEvent** function returns only when the specified predicate procedure returns a nonzero (true) for the next event in the queue that matches the specified event.

SEE ALSO

XFlush(3X), **XPutBackEvent**(3X)
Xlib – C Language X Interface

NAME

XInstallColormap, XUninstallColormap, XListInstalledColormaps – install colormaps

SYNTAX

XInstallColormap(*display, cmap*)
Display **display*;
Colormap *cmap*;

XUninstallColormap(*display, cmap*)
Display **display*;
Colormap *cmap*;

Colormap ***XListInstalledColormaps**(*display, w, num_return*)
Display **display*;
Window *w*;
int **num_return*;

OPTIONS

cmap	Specifies the colormap ID.
display	Specifies the connection to the X server.
num_return	Returns the list of currently installed colormaps.
w	Specifies the window ID.

DESCRIPTION

The **XInstallColormap** function installs the specified colormap for its associated screen. **XInstallColormap** can generate a **BadColor** error.

The **XUninstallColormap** function removes the specified colormap from the required list for its screen. **XUninstallColormap** can generate a **BadColor** error.

The **XListInstalledColormaps** function returns a list of the currently installed colormaps for the screen of the specified window. **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

Xlib – C Language X Interface

NAME

XInternAtom, XGetAtomName – create and 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*;

OPTIONS

atom	Specifies the atom associated with the string 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. You can pass either **True** or **False.**

DESCRIPTION

The **XInternAtom** function returns the atom identifier associated with the string you passed to the atom_name argument. **XInternAtom** can generate **BadAlloc** and **BadValue** errors.

The **XGetAtomName** function returns the name associated with the atom identifier you passed to the atom argument. You previously obtained the atom identifier by calling **XInternAtom. 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

XGetWindowProperty(3X)
Xlib – C Language X Interface

NAME

XIntersectRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XXorRegion, XOffsetRegion, XShrinkRegion, XPointInRegion, XRectInRegion – region arithmetic

SYNTAX

XIntersectRegion(*sra*, *srb*, *dr*)
Region *sra*, *srb*, *dr*;

XUnionRegion(*sra*, *srb*, *dr*)
Region *sra*, *srb*, *dr*;

XUnionRectWithRegion(*rectangle*, *src_region*, *dest_region*)
Rectangle **rectangle*;
Region *src_region*;
Region *dest_region*;

XSubtractRegion(*sra*, *srb*, *dr*)
Region *sra*, *srb*, *dr*;

XXorRegion(*sra*, *srb*, *dr*)
Region *sra*, *srb*, *dr*;

XOffsetRegion(*r*, *dx*, *dy*)
Region *r*;
int *dx*, *dy*;

XShrinkRegion(*r*, *dx*, *dy*)
Region *r*;
int *dx*, *dy*;

int **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*;

OPTIONS

dr	Stores the result of the computation.
dest_region	Specifies the destination region.
dx **dy**	Specifies the x and y coordinates.
r	Specifies the region.
rectangle	Specifies the rectangle.
sra **srb**	Specifies the two regions with which you want to perform the computation.
src_region	Specifies the source region to be used.
width **height**	Specifies the width and height.
x **y**	Specifies the x and y coordinates.

DESCRIPTION

The **XIntersectRegion** function computes the intersection of two regions.

The **XUnionRegion** function computes the union of two regions.

The **XUnionRectWithRegion** function creates the destination region from a union of the specified rectangle and the specified source region.

The **XSubtractRegion** function subtracts two regions.

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.

The **XPointInRegion** function returns nonzero 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(3X), **XEmptyRegion**(3X),
Xlib – C Language X Interface

NAME

XListFonts, XFreeFontNames – obtain and free font names

SYNTAX

char **XListFonts(*display, pattern, maxnames, actual_count_return*)
Display **display*;
char **pattern*;
int *maxnames*;
int **actual_count_return*;

XFreeFontNames(*list*)
char **list*[];

OPTIONS

actual_count_return	Returns the actual number of font names.
display	Specifies the connection to the X server.
list	Specifies the array of strings you want to free.
maxnames	Specifies the maximum number of names that are to be in the returned list.
pattern	Specifies the null-terminated string associated with the font names that you want returned. You can specify any string, an asterisk "*", or a question mark "?". The asterisk indicates a wildcard on any number of characters, and the question mark indicates a wildcard on a single character.

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 **XFreeFontNames** function frees the array and strings returned by **XListFonts.**

SEE ALSO

XLoadFont(3X), **XSetFontPath**(3X)
Xlib – C Language X Interface

NAME

XLoadFont, XQueryFont, XListFontsWithInfo, XFreeFontInfo, XLoadQueryFont, XFreeFont, XGetFontProperty, XUnloadFont, XGContextFromGC – manipulate fonts

SYNTAX

Font **XLoadFont**(*display*, *name*)
Display **display*;
char **name*;

XFontStruct ***XQueryFont**(*display*,*font_ID*)
Display **display*;
XID *font_ID*;

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*;

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*;

GContext **XGContextFromGC**(*gc*)
GC *gc*;

OPTIONS

actual_count	Must be the actual number of matched font names returned by **XListFontsWithInfo.**
atom	Specifies the atom associated with the string name you want returned.
count_return	Returns the actual number of matched font names.
display	Specifies the connection to the X server.
font	Specifies the font ID.
font_ID	Specifies the ID of the font or the graphics context.
gc	Specifies the graphics context that you want the resource for.
font_struct	Specifies the storage associated with the font.

info_return	Returns a pointer to the font information.
free_info	Must be the pointer to font information returned by **XListFontWithInfo.**
maxnames	Specifies the maximum number of names that are to be in the returned list.
name	Specifies the name of the font. This name is a null terminated string.
names	Must be the list of font names returned by **XListFontsWithInfo.**
pattern	Specifies the null-terminated string associated with the font names that you want returned. You can specify any string, an asterisk "*", or a question mark "?". The asterisk indicates a wildcard on any number of characters, and the question mark indicates a wildcard on a single character.
value_return	Returns the value of the font property.

DESCRIPTION

The **XLoadFont** function loads the specified font and returns its associated font ID. **XLoadFont** can generate **BadAlloc** and **BadName** errors.

The **XQueryFont** function returns a pointer to the **XFontStruct** structure, which contains information associated with the font. **XLoadQueryFont** can generate a **BadAlloc** error.

The **XListFontsWithInfo** function returns a list of names of fonts that match the specified pattern and their associated font information.

The **XFreeFontInfo** function frees the the font information array.

The **XLoadQueryFont** function provides the most common way for accessing a font. That is, **XLoadQueryFont** both opens (loads) the specified font and returns a pointer to the appropriate **XFontStruct** structure.

The **XFreeFont** 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 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. The function returns zero if the atom was not defined or one if it was defined.

The **XUnloadFont** function deletes the association between the font resource ID and the specified font. **XUnloadFont** can generate a **BadFont** error.

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

XListFonts(3X), **XSetFontPath**(3X)
Xlib – C Language X Interface

NAME

XLookupKeysym, XRefreshKeyboardMapping, XLookupString, XRebindKeySym – handle keyboard input events

SYNTAX

KeySym **XLookupKeysym**(*event_key*, *index*)
XKeyEvent **event_key*;
int *index*;

XRefreshKeyboardMapping(*event_map*)
XMappingEvent **event_map*;

int **XLookupString**(*event_struct*, *buffer_return*, *bytes_buffer*, *keysym_return*, *status_return*)
XKeyEvent **event_struct*;
char **buffer_return*;
int *bytes_buffer*;
KeySym **keysym_return*;
XComposeStatus **status_return*;

XRebindKeysym(*display*, *keysym*, *list*, *mod_count*, *string*, *bytes_string*)
Display **display*;
KeySym *keysym*;
KeySym **list*;
int *mod_count*;
unsigned char **string*;
int *bytes_string*;

OPTIONS

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_key	Specifies the key event that is to be used. This event is either a **KeyPress** event or a **KeyRelease** event.
event_map	Specifies the mapping event that is to be used.
event_struct	Specifies the key event structure to be used: **XKeyPressedEvent** or **XKeyReleasedEvent.**
index	Specifies the index into the KeySyms table.
keysym	Specifies the keysym to be rebound.
keysym_return	Returns the keysym computed from the event if this argument is not NULL.
list	Specifies a pointer to an array of keysyms that are being used as modifiers.
mod_count	Specifies the number of modifiers in the modifier list.
status_return	Specifies either a pointer to the **XCompose** structure that is to contain compose key state information and that allows compose key processing to take place, or **NULL.**
string	Specifies a pointer to the string that is to 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

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*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.

DESCRIPTION

The function maps the window and all of its subwindows which have had map requests. **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 which 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 to generate an **Expose** event on each newly displayed window. **XMapSubwindows** can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a window argument does not name a defined window.

SEE ALSO

XChangeWindowAttributes(3X), **XConfigureWindow**(3X), **XCreateWindow**(3X), **XDestroyWindow**(3X), **XRaiseWindow**(3X), **XUnmapWindow**(3X)
Xlib – C Language X Interface

NAME

XOpenDisplay, XCloseDisplay, XNoOp – connect or disconnect to X server

SYNTAX

Display ***XOpenDisplay**(*display_name*)
char **display_name*;

XCloseDisplay(*display*)
Display **display*;

XNoOp(*display*)
Display **display*;

OPTIONS

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 UNIX-based 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 the specified hardware display to the server through TCP, UNIX domain, or DECnet stream communications protocols.

The **XCloseDisplay** function closes the connection to the X server for the display specified in the **Display** structure.

The **XNoOp** function essentially sends a NoOperation request to the X server, thereby exercising the connection.

SEE ALSO

Xlib – C Language X Interface

NAME

XParseGeometry, XGeometry, 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*;

int **XGeometry**(*display*, *screen*, *position*, *default_position*, *bwidth*, *fwidth*, *fheight*, *xadder*, *yadder*, *x_return*, *y_return*, *width_return*, *height_return*)
Display **display*;
int *screen*;
char **position*, **default_position*;
unsigned int *bwidth*;
unsigned int *fwidth*, *fheight*;
int *xadder*, *yadder*;
int **x_return*, **y_return*;
int **width_return*, **height_return*;

Status **XParseColor**(*display*, *cmap*, *spec*, *exact_def_return*)
Display **display*;
Colormap *cmap*;
char **spec*;
XColor **exact_def_return*;

OPTIONS

bwidth	Specifies the border width.
cmap	Specifies the colormap ID.
display	Specifies the connection to the X server.
fheight **fwidth**	Specifies the font height and width in pixels (increment size).
parsestring	Specifies the string you want to parse.
position **default_position**	Specifies the geometry specifications.
screen	Specifies the screen.
exact_def_return	Returns the exact colors for later use and sets the **DoRed, DoGreen,** and **DoBlue** flags.
spec	Specifies the color name string. Uppercase and lowercase characters are acceptable.
width_return **height_return**	Returns the width and height determined.
xadder **yadder**	Specifies additional interior padding needed in the window.
x_return **y_return**	Returns the xoffset and yoffset determined.

DESCRIPTION

The **XParseGeometry** function parses standard window geometry strings.

The **XGeometry** function parses window geometry given an argument and a default position.

The **XParseColor** function parses color values. **XParseColor** can generate a **BadColor** error.

DIAGNOSTICS

BadColor A value for a colormap argument does not name a defined colormap.

SEE ALSO

Xlib – C Language X Interface

NAME

XPolygonRegion, XClipBox – generate regions

SYNTAX

Region **XPolygonRegion**(*points*, *n*, *fill_rule*)
XPoint *points[]*;
int *n*;
int *fill_rule*;

XClipBox(*r*, *rect*)
Region *r*;
XRectangle **rect*;

OPTIONS

fill_rule	Specifies the fill rule you want to set for the specified graphics context. You can pass one of these constants: **EvenOddRule** or **WindingRule.**
n	Specifies the number of points in the polygon.
points	Specifies an array of points.
r	Specifies the region.
rect	Specifies the rectangle.

DESCRIPTION

The **XPolygonRegion** function returns a region defined by the points array.

The **XClipBox** function generates the smallest enclosing rectangle in the specified rectangle that is located in the specified region.

SEE ALSO

Xlib – C Language X Interface

NAME

XPutBackEvent – put events back on the queue

SYNTAX

XPutBackEvent(*display*, *event*)
Display **display*;
XEvent **event*;

OPTIONS

display	Specifies the connection to the X server.
event	Specifies a pointer to the **XEvent** structure. This structure is a union of the individual structures declared for each event type.

DESCRIPTION

The **XPutBackEvent** function pushes an event back onto the head of the display's event queue.

SEE ALSO

XFlush(3X), **XIfEvent**(3X)
Xlib – C Language X Interface

NAME

XPutImage, XGetImage, XGetSubImage – transfer images

SYNTAX

XPutImage(*display, d, gc, image, src_x, src_y, dst_x, dst_y, width, height*)
Display **display*;
Drawable *d*;
GC *gc*;
XImage **image*;
int *src_x, src_y*;
int *dst_x, dst_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*;
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*;

OPTIONS

d Specifies the drawable.

dest_x
dest_y Specifies the x and y coordinates of the destination rectangle relative to its origin. These coordinates specify the upper-left corner of the destination rectangle. These coordinates determine where the subimage will be placed within the destination image.

display Specifies the connection to the X server.

dst_x
dst_y Specifies the x and y coordinates. These are the coordinates of the subimage and are relative to the origin of the drawable where the image will be drawn.

format Specifies the format for the image. You can pass one of these constants: **XYPixmap** or **ZPixmap.**

gc Specifies the graphics context.

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 from the top edge of the image defined by the **XImage** data

structure.

width
height Specifies the width and height of the subimage. These arguments define the dimensions of the rectangle.

x
y Specifies the x and y coordinates. These coordinates define the upper-left corner of the rectangle and are relative to the origin of the drawable.

DESCRIPTION

The **XPutImage** function combines an image in memory with a rectangle of the specified drawable. **XPutImage** can generate **BadDrawable, BadGC, BadMatch,** and **BadValue** errors.

The **XGetImage** function returns the **XImage** structure. This structure provides you with the contents of the specified rectangle of the drawable in the format you specify. **XGetImage** can generate **BadDrawable, BadMatch,** and **BadValue** errors.

The **XGetSubImage** function copies the contents of a rectangle in the specified drawable on the display to the specified location within a pre-existing image structure. **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

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*;

OPTIONS

class Specifies the class that you are interested in. You can pass one of these constants: **TileShape, CursorShape,** or **StippleShape.**

display Specifies the connection to the X server.

width
height Specifies the width and height.

which_screen Specifies any drawable on a screen.

width_return
height_return Returns 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. **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. 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. XQueryBestStipple** can generate **BadDrawable** and **BadMatch** errors.

DIAGNOSTICS

BadMatch An **InputOnly** window is used as a drawable.

BadDraw 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(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetState**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

NAME

XQueryColor, XQueryColors, XLookupColor – obtain color values

SYNTAX

XQueryColor (*display, cmap, def_return*)
Display **display*;
Colormap *cmap*;
XColor **def_return*;

XQueryColors (*display, cmap, defs_return, ncolors*)
Display **display*;
Colormap *cmap*;
XColor *defs_return*[];
int *ncolors*;

Status **XLookupColor** (*display, cmap, color_name, visual_def_return, exact_def_return*)
Display **display*;
Colormap *cmap*;
char **color_name*;
XColor **visual_def_return*, **exact_def_return*;

OPTIONS

cmap	Specifies the colormap ID.
def_in_out	Specifies or returns the RGB values for the pixel specified in the structure.
defs_in_out	Specifies or returns an array of color definition structures.
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.
exact_def_return	Returns the exact colors for later use and sets the **DoRed, DoGreen,** and **DoBlue** flags.

DESCRIPTION

The **XQueryColor** and **XQueryColors** functions returns the red, green, and blue color values stored in the specified colormap for the pixel value you pass in the pixel member of the **XColor** structure(s). The values returned for an unallocated entry are undefined. They also set the flags member in the **XColor** structure to all three colors. **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 cmap and returns both the exact color values and the closest values provided by the screen with respect to the visual type of the specified cmap.

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(3X), **XCreateColormap**(3X), **XStoreColors**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

child_return	Returns the child window ID 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 ID for the specified window.
root_x_return **root_y_return**	Returns the pointer coordinates relative to the root window's origin.
w	Specifies the window ID.
win_x_return **win_y_return**	Returns 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. **XQueryPointer** can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a window argument does not name a defined window.

SEE ALSO

XGetWindowAttributes(3X), **XQueryTree**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

children_return	Returns a pointer to the list of children for the specified window.
display	Specifies the connection to the X server.
nchildren_return	Returns the number of children for the specified window.
parent_return	Returns the parent window ID for the specified window.
root_return	Returns the root window ID for the specified window.
w	Specifies the window ID.

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. **XQueryTree** can generate a **BadWindow** error.

BUGS

This really should return a screen *, not a root window ID.

DIAGNOSTICS

BadWindow	A value for a window argument does not name a defined window.

SEE ALSO

XGetWindowAttributes(3X), **XQueryPointer**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

direction	Specifies the direction (up or down) that you want to circulate the window. You can pass one of these constants: **RaiseLowest** or **LowerHighest.**
display	Specifies the connection to the X server.
nwindows	Specifies the number of windows to be restacked.
w	Specifies the window ID.
windows	Specifies an array containing the windows to be restacked. All the windows must have the same parent.

DESCRIPTION

The **XRaiseWindow** function raises the specified window to the top of the stack so that no sibling window obscures it. **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. **XLowerWindow** can generate a **BadWindow** error.

The **XCirculateSubwindows** function circulates the specified subwindow in the specified direction: **RaiseLowest** or **LowerHighest. 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. **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. **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 will be unaffected, but the other windows in the array will be stacked underneath the first window in the order of the array. The stacking order of the other windows is not affected. **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(3X), **XConfigureWindow**(3X), **XCreateWindow**(3X), **XDestroyWindow**(3X), **XMapWindow**(3X), **XUnmapWindow**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

bitmap	Specifies the bitmap to be written.
bitmap_return	Returns the bitmap ID that is created.
d	Specifies the drawable.
data	Specifies the data in bitmap format.
data	Specifies the location of the bitmap data.
display	Specifies the connection to the X server.
depth	Specifies the depth of the pixmap.
fg **bg**	Specifies the foreground and background pixel values to use.
filename	Specifies the file name to use. The format of the file name is operating system specific.
width **height**	Specifies the width and height.
width_return **height_return**	Returns the width and height values of the read-in bitmap file.

x_hot
y_hot Specifies where to place the hot spot coordinates (or -1,-1 if none are present) in the file.

x_hot_return
y_hot_return Returns the hot spot coordinates.

DESCRIPTION

The **XReadBitmapFile** function reads in a file containing a bitmap.

The **XWriteBitmapFile** function writes a bitmap out to a file.

XCreatePixmapFromBitmapData creates a pixmap of the given depth and then does a bitmap-format **XPutImage** of the data into it.

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).

SEE ALSO

Xlib – C Language X Interface

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*;

OPTIONS

background_color	Specifies the red, green, and blue (RGB) values for the background of the source.
cursor	Specifies the cursor.
d	Specifies the drawable.
display	Specifies the connection to the X server.
foreground_color	Specifies the red, green, and blue (RGB) values for the foreground of the source.
width **height**	Specifies the width and height of the cursor that you want the size information for.
width_return **height_return**	Returns 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. **XFreeCursor** can generate a **BadCursor** error.

The **XQueryBestCursor** function returns the closest shape actually supported by the display hardware. **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

XCreateFontCursor(3X), **XDefineCusor**(3X)
Xlib – C Language X Interface

NAME

XReparentWindow – reparent windows

SYNTAX

XReparentWindow(*display*, *w*, *parent*, *x*, *y*)
Display **display*;
Window *w*;
Window *parent*;
int *x*, *y*;

OPTIONS

display	Specifies the connection to the X server.
parent	Specifies the parent window ID.
w	Specifies the window ID.
x **y**	Specifies the x and y coordinates of the position in the new parent window of the specified window.

DESCRIPTION

The **XReparentWindow** function reparents the specified window by inserting it as the child of the specified parent. **XReparentWindow** can generate **BadMatch** and **BadWindow** errors.

DIAGNOSTICS

BadWindow A value for a window argument does not name a defined window.

SEE ALSO

XChangeSaveSet(3X)
Xlib – C Language X Interface

NAME

XSaveContext, XFindContext, XDeleteContext, XUniqueContext – manipulate the context manager

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**()

OPTIONS

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

The **XSaveContext** function saves a data value that corresponds to a window and context type.

The **XFindContext** function gets the data associated with a window and type.

The **XDeleteContext** function deletes an entry for a given window and type.

The **XUniqueContext** function creates a unique context type that may be used in subsequent calls to **XSaveContext.**

SEE ALSO

Xlib – C Language X Interface

NAME

XSelectInput – select input events

SYNTAX

XSelectInput(*display*, *w*, *event_mask*)
Display **display*;
Window *w*;
unsigned long *event_mask*;

OPTIONS

display	Specifies the connection to the X server.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
w	Specifies the window ID.

DESCRIPTION

The **XSelectInput** function requests that the X server report the events associated with the event masks that you pass to the **event_mask** argument. **XSelectInput** 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

Xlib – C Language X Interface

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*;
Boolean *graphics_exposures*;

OPTIONS

arc_mode	Specifies the arc mode: **ArcChord** specifies that arcs will be chord filled, while **ArcPieSlice** specifies that arcs will be pie slice filled.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
graphics_exposures	Specifies whether you want **GraphicsExpose** events to be reported when calling **XCopyArea** and **XCopyPlane** with this graphics context. If **True, GraphicsExpose** events are reported. If **False, GraphicsExpose** events are not reported.
subwindow_mode	Specifies the subwindow mode: **ClipByChildren** clips source and destination by all viewable children, while **IncludeInferiors** draws through all subwindows, that is, does not clip by inferiors.

DESCRIPTION

The **XSetArcMode** function sets the arc mode in the specified graphics context. **XSetArcMode** can generate **BadGC** and **BadValue** errors.

The **XSetSubwindowMode** function sets the subwindow mode in the specified graphics context. **XSetSubwindowMode** can generate **BadGC** and **BadValue** errors.

The **XSetGraphicsExposures** function sets the graphics-exposures flag in the specified graphics context. **XSetGraphicsExposures** can generate **BadGC** and **BadValue** errors.

DIAGNOSTICS

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(3X), **XQueryBestSize**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetState**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

NAME

XSetClassHint, XGetClassHint – set or get class hint

SYNTAX

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*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.
class_hints	Specifies a pointer to a **XClassHint** structure that is to be used.
class_hints_return	Returns the **XClassHints** structure.

DESCRIPTION

The **XSetClassHint** functions sets the class hint for the specified window. **XSetClassHint** can generate **BadAlloc** and **BadWindow** errors.

The **XGetClassHint** function obtains the class of the specified window. **XGetClassHint** can generate a **BadWindow** error.

PROPERTY

WM_CLASS

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

XSetCommand(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

display	Specifies the connection to the X server.
clip_x_origin **clip_y_origin**	Specifies the x and y coordinates of the clip origin.
gc	Specifies the graphics context.
n	Specifies the number of rectangles.
ordering	Specifies the ordering relations on the rectangles. Possible values are **Unsorted, YSorted, YXSorted,** or **YXBanded.**
pixmap	Specifies the pixmap.
rectangles	Specifies an array of rectangles.

DESCRIPTION

The **XSetClipOrigin** function sets the clip origin in the specified graphics context. **XSetClipOrigin** can generate a **BadGC** error.

The **XSetClipMask** function sets the **clip_mask** in the specified graphics context to the specified pixmap. **XSetClipMask** can generate **BadGC, BadMatch,** and **BadValue** errors.

The **XSetClipRectangles** function changes the **clip_mask** in the specified graphics context to the specified list of rectangles and sets the clip origin. **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(3X), **XQueryBestSize**(3X), **XSetArcMode**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetState**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

NAME

XSetCloseDownMode, XKillClient – control clients

SYNTAX

XSetCloseDownMode(*display, close_mode*)
Display **display*;
int *close_mode*;

XKillClient(*display, resource*)
Display **display*;
XID *resource*;

OPTIONS

close_mode	Specifies the client close down mode you want to change. You can pass one of these constants: **DestroyAll, RetainPermanent,** or **RetainTemporary.**
display	Specifies the connection to the X server.
resource	Specifies any resource associated with the client you want to destroy. You can also pass **AllTemporary.**

DESCRIPTION

The **XSetCloseDownMode** defines what will happen to the client's resources at connection close. **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. **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

NAME

XSetCommand – set command atom

SYNTAX

XSetCommand(*display*, *w*, *argv*, *argc*)
Display **display*;
Window *w*;
char ***argv*;
int *argc*;

OPTIONS

argc	Specifies the number of arguments.
argv	Specifies a pointer to the command and arguments used to start the application.
display	Specifies the connection to the X server.
w	Specifies the window ID.

DESCRIPTION

The **XSetCommand** function records the command and arguments used to invoke the application. **XSetCommand** can generate **BadAlloc** and **BadWindow** errors.

PROPERTY

WM_COMMAND

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

XSetClassHint(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

NAME

XSetErrorHandler, XGetErrorText, XGetErrorDatabaseText, XDisplayName, XSetIOErrorHandler – default error handlers

SYNTAX

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*;

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	Specifies the buffer that you want the error message stored in.
code	Specifies the error code for which you want to obtain a description.
default_string	Specifies the default error message.
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.
nbytes	Specifies the number of bytes available in the buffer.
string	Specifies the character string.

DESCRIPTION

The **XSetErrorHandler** function handles error events.

The **XGetErrorText** function copies a null-terminated string describing the specified error code into the specified buffer.

The **XDisplayName** function returns the name of the display that you are currently using.

The **XSetIOErrorHandler** sets the fatal IO error handler.

The **XGetErrorDatabaseText** function returns a message (or the default message) from the error message database.

SEE ALSO

XSynchronize(3X)
Xlib – C Language X Interface

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*;

OPTIONS

display	Specifies the connection to the X server.
fill_rule	Specifies the fill rule you want to set for the specified graphics context. You can pass one of these constants: **EvenOddRule** or **WindingRule.**
fill_style	Specifies the fill style you want to set for the specified graphics context. Possible values are **FillSolid, FillTiled, FillStippled,** or **FillOpaqueStippled.**
gc	Specifies the graphics context.

DESCRIPTION

The **XSetFillStyle** function sets the fill style in the specified graphics context. **XSetFillStyle** can generate **BadGC** and **BadValue** errors.

The **XSetFillRule** function sets the fill rule in the specified graphics context. **XSetFillRule** can generate **BadGC** and **BadValue** errors.

DIAGNOSTICS

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(3X), **XQueryBestSize**(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetState**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

NAME

XSetFont – GC convenience routines

SYNTAX

XSetFont(*display*, *gc*, *font*)
Display **display*;
GC *gc*;
Font *font*;

OPTIONS

display	Specifies the connection to the X server.
font	Specifies the font ID.
gc	Specifies the graphics context.

DESCRIPTION

The **XSetFont** function sets the current font in the specified graphics context. **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(3X), **XQueryBestSize**(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetLineAttributes**(3X), **XSetState**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

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. **XSetFontPath** can generate a **BadValue** error.

The **XGetFontPath** function allocates and returns an array of strings containing the search path. The data in the font path should be freed when no longer needed.

The **XFreeFontPath** function, when presented the data from **XGetFontPath,** frees the data used by the array.

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(3X), **XLoadFonts**(3X)
Xlib – C Language X Interface

NAME

XSetIconName, XGetIconName – set or get icon names

SYNTAX

XSetIconName(*display, w, icon_name*)
Display **display*;
Window *w*;
char **icon_name*;

int **XGetIconName**(*display, w, icon_name_return*)
Display **display*;
Window *w*;
char ***icon_name_return*;

OPTIONS

display	Specifies the connection to the X server.
icon_name	Specifies the name to be displayed in the window's icon.
icon_name_return	Returns a pointer to the name to be displayed in the window's icon. The name will be a null-terminated string.
w	Specifies the window ID.

DESCRIPTION

The **XSetIconName** function sets the name to be displayed in a window's icon. **XSetIconName** can generate **BadAlloc** and **BadWindow** errors.

The **XGetIconName** function obtains the window name to be displayed in its icon and either returns a nonzero if it succeeds or zero if it fails (for example, if no icon name has been set for the argument window). **XGetIconName** can generate a **BadWindow** error.

PROPERTY

WM_ICON_NAME

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

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

NAME

XSetIconSizes, XGetIconSizes – set or get icon size hints

SYNTAX

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*;

OPTIONS

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 ID.

DESCRIPTION

The **XSetIconSizes** function sets the value of the icon size atom. **XSetIconSizes** can generate **BadAlloc** and **BadWindow** errors.

The **XGetIconSizes** function returns the value of the icon sizes atom. **XGetIconSizes** can generate a **BadWindow** error.

PROPERTY

WM_ICON_SIZE

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

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconName**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

display	Specifies the connection to the X server.
focus	Specifies the window ID.
focus_return	Returns the focus window ID, or either **PointerRoot** or **None.**
revert_to	Specifies which window the input focus reverts to if the window becomes not viewable. You can pass one of these constants: **RevertToParent, RevertToPointerRoot,** or **RevertToNone.**
revert_to_return	Returns the current focus state. The function can return one of these constants: **RevertToParent, RevertToPointerRoot,** or **RevertToNone.**
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or **CurrentTime.**

DESCRIPTION

The **XSetInputFocus** function changes the input focus and the last-focus-change time. **XSetInputFocus** can generate **BadMatch, BadValue,** and **BadWindow** errors.

The **XGetInputFocus** function returns the focus window ID 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(3X)
Xlib – C Language X Interface

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*;

OPTIONS

cap_style	Specifies the line and cap style you want to set for the specified graphics context. Possible values are **CapNotLast, CapButt, CapRound,** or **CapProjecting.**
dash_list	Specifies the dash list for the dashed line style you want to set for the specified graphics context.
dash_offset	Specifies the phase of the pattern for the dashed line style you want to set for the specified graphics context.
display	Specifies the connection to the X server.
gc	Specifies the graphics context.
join_style	Specifies the line-join style you want to set for the specified graphics context. Possible values are **JoinMiter, JoinRound,** or **JoinBevel.**
line_style	Specifies the line style you want to set for the specified graphics context. Possible values are **LineSolid** (solid), **LineOnOffDash** (on-off dash), or **LineDoubleDash** (double dash).
line_width	Specifies the line width you want to set for the specified graphics context.
n	Specifies the number of elements in the dash list argument.

DESCRIPTION

The **XSetLineAttributes** function sets the line drawing components in the specified graphics context. **XSetLineAttributes** can generate **BadGC** and **BadValue** errors.

The **XSetDashes** function sets the **dash_offset** and **dash_list** for dashed line styles in the specified graphics context. **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(3X), **XQueryBestSize**(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetState**(3X), **XSetTile**(3X)

Xlib – C Language X Interface

NAME

XSetNormalHints, XGetNormalHints – set or get normal state hints

SYNTAX

void **XSetNormalHints**(*display*, *w*, *hints*)
Display **display*;
Window *w*;
XSizeHints **hints*;

Status **XGetNormalHints**(*display*, *w*, *hints_return*)
Display **display*;
Window *w*;
XSizeHints **hints_return*;

OPTIONS

display	Specifies the connection to the X server.
hints	Specifies a pointer to the sizing hints for the window in its normal state.
hints_return	Returns the sizing hints for the window in its normal state.
w	Specifies the window ID.

DESCRIPTION

The **XSetNormalHints** function sets the size hints for a window in its normal state. **XSetNormalHints** can generate **BadAlloc** and **BadWindow** errors.

The **XGetNormalHints** function returns the size hints for a window in its normal state. **XGetNormalHints** can generate a **BadWindow** error.

PROPERTY

WM_NORMAL_HINTS

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

XSetCommand(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

NAME

XSetPointerMapping, XGetPointerMapping – manipulate pointer settings

SYNTAX

int **XSetPointerMapping**(*display*, *map*, *nmap*)
Display **display*;
unsigned char *map*[];
int *nmap*;

int **XGetPointerMapping**(*display*, *map*, *nmap*)
Display **display*;
unsigned char *map*[];
int *nmap*;

OPTIONS

display	Specifies the connection to the X server.
map	Specifies the mapping list.
nmap	Specifies the number of items in mapping list.

DESCRIPTION

The **XSetPointerMapping** function sets the mapping of the pointer and causes the X server to generate a **MappingNotify** event on a status of **MappingSuccess. XSetPointerMapping** can generate a **BadValue** error.

The **XGetPointerMapping** function returns the current mapping of the pointer.

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 arguments type is accepted. Any argument defined as a set of alternatives can generate this error.

SEE ALSO

XChangeKeyboardControl(3X), **XChangeKeyboardMapping**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

allow_exposures	Specifies the current screen save control values. Possible values are **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 between screen saver invocations.
interval_return	Returns the interval between screen saver invocations.
mode	Specifies the mode that is to be applied.
prefer_blanking	Specifies whether to enable screen blanking. Possible values are **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 minutes, until the screen saver turns on.

DESCRIPTION

The **XSetScreenSaver** function sets the screen saver. **XSetScreenSaver** can generate a **BadValue** error.

The **XForceScreenSaver** function forces the screen saver. **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

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*;

OPTIONS

display	Specifies the connection to the X server.
owner	Specifies the owner of the specified selection atom. You can pass a window ID or **None.**
property	Specifies the property atom.
requestor	Specifies the requestor.
selection	Specifies the selection atom.
target	Specifies the target atom.
time	Specifies the time. You can pass either a timestamp, expressed in milliseconds, or **CurrentTime.**

DESCRIPTION

The **XSetSelectionOwner** function changes the owner and last change time for the specified selection. **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.** **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 event. **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

NAME

XSetSizeHints, XGetSizeHints – set or get window size hints

SYNTAX

XSetSizeHints(*display, w, hints, property*)
Display **display*;
Window *w*;
XSizeHints **hints*;
Atom *property*;

Status **XGetSizeHints**(*display, w, hints_return, property*)
Display **display*;
Window *w*;
XSizeHints **hints_return*;
Atom *property*;

OPTIONS

display	Specifies the connection to the X server.
hints	Specifies a pointer to the size hints.
hints_return	Returns the size hints.
property	Specifies the property atom.
w	Specifies the window ID.

DESCRIPTION

The **XSetSizeHints** function sets the value of any property of type WM_SIZE_HINTS. **XSetSizeHints** can generate **BadAlloc, BadAtom,** and **BadWindow** errors.

The **XGetSizeHints** function reads the value of any property of type WM_SIZE_HINTS. **XGetSizeHints** 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.
BadWindow	A value for a window argument does not name a defined window.

SEE ALSO

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

NAME

XSetStandardColormap, XGetStandardColormap – set or get standard colormaps

SYNTAX

void **XSetStandardColormap**(*display, w, cmap, property*)
Display **display*;
Window *w*;
XStandardColormap **cmap*;
Atom *property*; /* RGB_BEST_MAP, etc. */

Status **XGetStandardColormap**(*display, w, cmap_return, property*)
Display **display*;
Window *w*;
XStandardColormap **cmap_return*;
Atom *property*; /* RGB_BEST_MAP, etc. */

OPTIONS

cmap	Specifies the colormap ID.
cmap_return	Returns the colormap associated with the specified atom.
display	Specifies the connection to the X server.
property	Specifies the property atom.
w	Specifies the window ID.

DESCRIPTION

The **XSetStandardColormap** function creates or changes a standard colormap. **XSetStandardColormap** can generate **BadAlloc, BadAtom,** and **BadWindow** errors.

The **XGetStandardColormap** function gets the **XStandardColormap** structure associated with one of the described atoms. **XGetStandardColormap** 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.
BadWindow	A value for a window argument does not name a defined window.

SEE ALSO

Xlib – C Language X Interface

NAME

XSetStandardProperties – set standard window manager properties

SYNTAX

XSetStandardProperties (*display*, *w*, *window_name*, *icon_name*, *icon_pixmap*, *argv*, *argc*, *hints*)
Display **display*;
Window *w*;
char **window_name*;
char **icon_name*;
Pixmap *icon_pixmap*;
char ***argv*;
int *argc*;
XSizeHints **hints*;

OPTIONS

argc	Specifies the number of arguments.
argv	Specifies a pointer to the command and arguments used to start the application.
display	Specifies the connection to the X server.
hints	Specifies a pointer to the sizing hints for the window in its normal state.
icon_name	Specifies the name to be displayed in the window's icon.
icon_pixmap	Specifies the single plane pixmap that is to be used for the icon or **None.**
w	Specifies the window ID.
window_name	Specifies the name of the window.

DESCRIPTION

The **XSetStandardProperties** function provides a means by which, with a single call, simple applications set the most essential properties. **XSetStandardProperties** can generate **BadAlloc** and **BadWindow** errors.

PROPERTIES

WM_NAME, WM_ICON_NAME, WM_HINTS, WM_COMMAND, and WM_NORMALHINTS

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

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

background	Specifies the background you want to set for the specified graphics context.
display	Specifies the connection to the X server.
foreground	Specifies the foreground you want to set for the specified graphics context.
function	Specifies the function you want to set for the specified graphics context.
gc	Specifies the graphics context.
plane_mask	Specifies the plane mask.

DESCRIPTION

The **XSetState** function sets the foreground, background, plane mask, and function components for the specified graphics context. **XSetState** can generate **BadGC** and **BadValue** errors.

XSetFunction sets a specified value in the specified graphics context. **XSetFunction** can generate **BadGC** and **BadValue** errors.

The **XSetPlaneMask** function sets the plane mask in the specified graphics context. **XSetPlaneMask** can generate a **BadGC** error.

The **XSetForeground** function sets the foreground in the specified graphics context. **XSetForeground** can generate a **BadGC** error.

The **XSetBackground** function sets the background in the specified graphics context. **XSetBackground** can generate a **BadGC** error.

DIAGNOSTICS

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(3X), **XQueryBestSize**(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetTile**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

display	Specifies the connection to the X server.
gc	Specifies the graphics context.
stipple	Specifies the stipple you want to set for the specified graphics context.
tile	Specifies the fill tile you want to set for the specified graphics context.
ts_x_origin **ts_y_origin**	Specifies the x and y coordinates of the tile or stipple origin.

DESCRIPTION

The **XSetTile** function sets the fill tile in the specified graphics context. **XSetTile** can generate **BadAlloc, BadGC, BadMatch,** and **BadPixmap** errors.

The **XSetStipple** function sets the stipple in the specified graphics context. **XSetStipple** can generate **BadAlloc, BadGC, BadMatch,** and **BadPixmap** errors.

The **XSetTSOrigin** function sets the tile/stipple origin in the specified graphics context. **XSetTSOrigin** can generate a **BadGC** error.

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(3X), **XQueryBestSize**(3X), **XSetArcMode**(3X), **XSetClipOrigin**(3X), **XSetFillStyle**(3X), **XSetFont**(3X), **XSetLineAttributes**(3X), **XSetState**(3X)
Xlib – C Language X Interface

NAME

XSetTransientForHint, XGetTransientForHint – set or get transient for hint

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*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.
prop_window	Specifies the window ID 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** set the WM_TRANSIENT_FOR atom of the specified window to the specified **prop_window. XSetTransientForHint** can generate **BadAlloc** and **BadWindow** errors.

The **XGetTransientForHint** function obtains the WM_TRANSIENT_FOR property for the specified window. **XGetTransientForHint** can generate a **BadWindow** error.

PROPERTY

WM_TRANSIENT_FOR

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

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

NAME

XSetWMHints, XGetWMHints – set or get window manager hints

SYNTAX

XSetWMHints(*display, w, wmhints*)
Display **display*;
Window *w*;
XWMHints **wmhints*;

XWMHints ***XGetWMHints**(*display, w*)
Display **display*;
Window *w*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.
wmhints	Specifies a pointer to the window manager hints.

DESCRIPTION

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 value of the window manager hints atom and returns either NULL if it fails (for example, if no WM_HINTS property was set on window w) or a pointer to a XWMHints structure if it succeeds. **XGetWMHints** can generate a **BadWindow** error.

PROPERTY

WM_HINTS

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

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetZoomHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

NAME

XSetZoomHints, XGetZoomHints – set or get zoom state hints

SYNTAX

XSetZoomHints(*display, w, zhints*)
Display **display*;
Window *w*;
XSizeHints **zhints*;

Status **XGetZoomHints**(*display, w, zhints_return*)
Display **display*;
Window *w*;
XSizeHints **zhints_return*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.
zhints	Specifies a pointer to the zoom hints.
zhints_return	Returns the zoom hints.

DESCRIPTION

The **XSetZoomHints** function sets the value of the zoom hints atom. **XSetZoomHints** can generate **BadAlloc** and **BadWindow** errors.

The **XGetZoomHints** function returns the value of the zoom hints atom. **XGetZoomHints** can generate a **BadWindow** error.

PROPERTY

WM_ZOOM_HINTS

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

XSetClassHint(3X), **XSetCommand**(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetTransientForHint**(3X), **XSetWMHints**(3X), **XStoreName**(3X)
Xlib – C Language X Interface

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*, *return_buffer*)
Display **display*;
int **nbytes_return*;
int *return_buffer*;

XRotateBuffers(*display*, *rotate*)
Display **display*;
int *rotate*;

OPTIONS

buffer	Specifies the buffer in which you want to store the byte string.
bytes	Specifies the string of bytes you want stored. The byte string is not necessarily ASCII or null-terminated.
display	Specifies the connection to the X server.
nbytes	Specifies the number of bytes of the bytes argument that you want stored.
nbytes_return	Returns the number of bytes in the string to the buffer.
return_buffer	Specifies which buffer you want the stored data to be returned from.
rotate	Specifies how much to rotate the cut buffer.

DESCRIPTION

The **XStoreBytes** function returns the number of bytes to be stored to the nbytes argument. **XStoreBytes** can generate **BadAlloc** and **BadWindow** errors.

The **XStoreBuffer** function stores data in a specified cut buffer. **XStoreBuffer** can generate **BadAlloc, BadAtom,** and **BadWindow** errors.

The **XFetchBytes** function returns the number of bytes in the nbytes argument, if the buffer contains data. Otherwise, the function returns NULL and sets nbytes to 0. **XFetchBytes** can generate a **BadWindow** error.

The **XFetchBuffer** function returns the value zero to the nbytes 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. **XRotateBuffers** can generate **BadAtom, BadMatch,** 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.

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

Xlib – C Language X Interface

NAME

XStoreColors, XStoreColor, XStoreNamedColor – set colors

SYNTAX

XStoreColors(*display*, *cmap*, *color*, *ncolors*)
Display **display*;
Colormap *cmap*;
XColor *color*[];
int *ncolors*;

XStoreColor(*display*, *cmap*, *color*)
Display **display*;
Colormap *cmap*;
XColor **color*;

XStoreNamedColor(*display*, *cmap*, *color*, *pixel*, *flags*)
Display **display*;
Colormap *cmap*;
char **color*;
unsigned long *pixel*;
int *flags*;

OPTIONS

cmap	Specifies the colormap ID.
color	Specifies the pixel and RGB values.
color	Specifies an array of color definition structures to be stored.
display	Specifies the connection to the X server.
flags	Specifies which red, green, and blue indexes are set.
ncolors	Specifies the number of **XColor** structures in the color definition array.
pixel	Specifies the entry in the colormap.
exact_def_return	Returns the exact colors for later use and sets the **DoRed, DoGreen,** and **DoBlue** flags.

DESCRIPTION

The **XStoreColors** function changes the colormap entries of the pixel values specified in the pixel members of the **XColor** structures. **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. **XStoreColor** can generate **BadColor** and **BadValue** errors.

The **XStoreNamedColor** function looks up the named color with respect to the screen associated with cmap and stores the result in cmap. **XStoreNamedColor** can generate **BadAccess, BadColor, BadName,** and **BadValue** errors.

DIAGNOSTICS

BadAccess	A client attempted to free a colormap entry that it did not already allocate.
BadAccess	A client attempted to store into a read-only colormap 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(3X), **XCreateColormap**(3X), **XQueryColor**(3X)
Xlib – C Language X Interface

NAME

XStoreName, XFetchName – set or get window names

SYNTAX

XStoreName(*display*, *w*, *window_name*)
Display **display*;
Window *w*;
char **window_name*;

int **XFetchName**(*display*, *w*, *window_name_return*)
Display **display*;
Window *w*;
char ***window_name_return*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.
window_name	Specifies the name of the window.
window_name_return	Returns a pointer to the window name, which will be a null-terminated string.

DESCRIPTION

The **XStoreName** function assigns the name passed to **window_name** to the specified window. **XStoreName** can generate **BadAlloc** and **BadWindow** errors.

The **XFetchName** function obtains a window name and returns either a nonzero if it succeeds or zero if it fails (for example, if no name has been set for the argument window). **XFetchName** can generate a **BadWindow** error.

PROPERTY

WM_NAME

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

XSetCommand(3X), **XSetIconName**(3X), **XSetIconSizeHints**(3X), **XSetNormalHints**(3X), **XSetSizeHints**(3X), **XSetStandardProperties**(3X), **XSetWMHints**(3X), **XSetZoomHints**(3X)
Xlib – C Language X Interface

NAME

XStringToKeysym, XKeysymToString, XKeycodeToKeysym, XKeysymToKeycode – convert keysyms

SYNTAX

KeySym **XStringToKeysym**(*string*)
char **string*;

char **XKeysymToString*(*keysym_str*)
KeySym *keysym_str*;

KeySym **XKeycodeToKeysym**(*display*, *keycode*, *index_return*)
Display **display*;
KeyCode *keycode*;
int *index_return*;

KeyCode **XKeysymToKeycode**(*display*, *keysym_kcode*)
Display **display*;
Keysym *keysym_kcode*;

OPTIONS

display	Specifies the connection to the X server.
index_return	Returns the element of keycode vector.
keycode	Specifies the keycode.
keysym_kcode	Specifies the keysym that is to be searched for.
keysym_str	Specifies the keysym that is to be converted.
string	Specifies the name of the keysym that is to be converted.

DESCRIPTION

The **XStringToKeysym** function converts the name of the keysym to the keysym code.

The **XKeysymToString** function converts a keysym code to the name of the keysym.

The **XKeycodeToKeysym** function converts a keycode to a defined keysym.

The **XKeysymToKeycode** function converts a keysym to the appropriate keycode.

SEE ALSO

XLookupKeysym(3X)
Xlib – C Language X Interface

NAME

XSynchronize, XSetAfterFunction – enable or disable synchronization

SYNTAX

int (***XSynchronize**(*display, onoff*))()
Display **display*;
int *onoff*;

int (***XSetAfterFunction**(*display, proc*))()
Display **display*;
int (**proc*)();

OPTIONS

display	Specifies the connection to the X server.
proc	Specifies the function to be called after an Xlib function that generates a protocol request completes its work.
onoff	Specifies whether to enable or disable synchronization. Possible values you can pass are 0 (disable synchronization) or nonzero (enable synchronization).

DESCRIPTION

The **XSynchronize** function enables or disables synchronization.

After completing their work, all Xlib functions that generate protocol requests call what is known as a previous after function. **XSetAfterFunction** sets which function is to be called.

SEE ALSO

XSetErrorHandler(3X)
Xlib – C Language X Interface

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;
```

OPTIONS

font_ascent_return	Returns the font ascent member.
font_descent_return	Returns the font descent member.
direction_return	Returns the value of the direction hint member: **FontLeftToRight** or **FontRightToLeft.**
display	Specifies the connection to the X server.
font_ID	Specifies either the font ID or the graphics context that contains the font.
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 return the logical extents of the specified 1-byte and 2-byte character string.

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. **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

XTextWidth(3X)
Xlib – C Language X Interface

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*;

OPTIONS

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 strings, ignoring size bearings.

SEE ALSO

XTextExtents(3X)
Xlib – C Language X Interface

NAME

XTranslateCoordinates – translate window coordinates

SYNTAX

int **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*;

OPTIONS

child_return	Returns the child if the coordinates are contained in a mapped child of the destination window.
dest_w	Specifies the window ID of the destination window.
dest_x_return **dest_y_return**	Returns the x and y coordinates within the destination window.
display	Specifies the connection to the X server.
src_w	Specifies the window ID of 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 within the source window 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 the child argument. **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

NAME

XUnmapWindow, XUnmapSubwindows – unmap windows

SYNTAX

XUnmapWindow(*display, w*)
Display **display*;
Window *w*;

XUnmapSubwindows(*display, w*)
Display **display*;
Window *w*;

OPTIONS

display	Specifies the connection to the X server.
w	Specifies the window ID.

DESCRIPTION

The **XUnmapWindow** function unmaps the specified window and causes the X server to generate an **UnmapNotify** event. **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 exposure events on formerly obscured windows. **XUnmapSubwindows** can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a window argument does not name a defined window.

SEE ALSO

XChangeWindowAttributes (3X), XConfigureWindow (3X), XCreateWindow (3X), XDestroyWindow (3X), XMapWindow (3X) XRaiseWindow (3X)
Xlib – C Language X Interface

NAME

XWarpPointer – control input focus

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*;

OPTIONS

dest_w Specifies the window ID of the destination window.

dest_x
dest_y Specifies the x and y coordinates within the destination window.

display Specifies the connection to the X server.

src_width
src_height Specifies the width and height of the source window.

src_w Specifies the window ID of the source window.

src_x
src_y Specifies the x and y coordinates within the source window.

DESCRIPTION

The **XWarpPointer** function moves the pointer to the coordinates specified by the **dest_x** and **dest_y** arguments, relative to the destination window's origin. If the destination window is **None**, the pointer is moved by offsets specified by the **dest_x** and **dest_y** coordinates. **XWarpPointer** can generate a **BadWindow** error.

DIAGNOSTICS

BadWindow A value for a window argument does not name a defined window.

SEE ALSO

XSetInputFocus(3X)
Xlib – C Language X Interface

NAME

XWindowEvent, XCheckWindowEvent, XMaskEvent, XCheckMaskEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XGetMotionEvents, XSendEvent – select event types

SYNTAX

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*;
int *event_mask*;
XEvent **event_return*;

XMaskEvent(*display*, *event_mask*, *event_return*)
Display **display*;
unsigned long *event_mask*;
XEvent **event_return*;

Bool **XCheckMaskEvent**(*display*, *event_mask*, *event_return*)
Display **display*;
unsigned 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*;

XTimeCoord ***XGetMotionEvents**(*display*, *w*, *start*, *stop*, *nevents_return*)
Display **display*;
Window *w*;
Time *start*, *stop*;
int **nevents_return*;

Status **XSendEvent**(*display*, *w*, *propagate*, *event_mask*, *event_send*)
Display **display*;
Window *w*;
Bool *propagate*;
unsigned long *event_mask*;
XEvent **event_send*;

OPTIONS

display	Specifies the connection to the X server.
event_mask	Specifies the event mask. This mask is the bitwise inclusive OR of one or more of the valid event mask bits.
event_return	Copies the matched event's associated structure into this client-supplied structure.
event_send	Specifies a pointer to the event that is to be sent.

event_type	Specifies the event type to be compared.
nevents_return	Returns the number of events from the motion history buffer.
propagate	Specifies a boolean value that is either **True** or **False.**
start **stop**	Specifies the time interval in which the events are returned from the motion history buffer. You can pass a time stamp, expressed in milliseconds, or **CurrentTime.** If the stop time is in the future, it is equivalent to specifying **CurrentTime.**
w	Specifies the window ID.

DESCRIPTION

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 **XCheckWindowEvent** function searches the event queue, then the events available on the server connection, for the first event that matches the specified window and event mask. When it finds a match, **XCheckWindowEvent** removes that event, copies it into the specified **XEvent** structure, and returns **True.**

The **XMaskEvent** function searches the event queue for the events associated with the specified mask. When it finds a match, **XMaskEvent** removes that and copies it into the specified **XEvent** structure.

The **XCheckMaskEvent** function searches first the event queue, then any events available on the server connection, for the first event that matches the specified mask. When it finds a match, **XCheckMaskEvent** removes that event, copies it into the specified **XEvent** structure, and returns **True.**

The **XCheckTypedEvent** function searches first the event queue, then any events available on the server connection, for the first event that matches the specified type. When it finds a match, **XCheckTypedEvent** returns its associated event structure to the specified **XEvent** structure and returns **True.**

The **XCheckTypedWindowEvent** function searches first the event queue, then any events available on the server connection, for the first event that matches the specified type and window. When it finds a match, **XCheckTypedWindowEvent** removes the event from the queue, copies it into the specified **XEvent** structure and returns **True.**

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 (including borders) the specified window at its present placement. **XGetMotionEvents** can generate a **BadWindow e** error.

The **XSendEvent** function sends an event to a specified window. **XSendEvent** can generate **BadValue** and **BadWindow e** 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

Xlib – C Language X Interface

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*;

Bool **XrmQGetSearchList**(*database*, *names*, *classes*, *list_return*, *list_length*)
XrmDatabase *database*;
XrmNameList *names*;
XrmClassList *classes*;
XrmSearchList *list_return*;
int *list_length*;

Bool **XrmQGetSearchResource**(*list*, *names*, *classes*, *type_return*, *value_return*)
XrmSearchList *list*;
XrmName *names*;
XrmClass *classes*;
XrmRepresentation **type_return*;
XrmValue **value_return*;

OPTIONS

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.
str_class	Specifies the fully qualified class of the value being retrieved (as a string).
names	Specifies a list of resource names.
str_type_return	Returns a pointer to the representation type of the destination (as a string).
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_name	Specifies the fully qualified name of the value being retrieved (as a string).
value_return	Returns the value in the database.

DESCRIPTION

The **XrmGetResource** and **XrmQGetResource** functions retrieve a resource from the specified database.

The **XrmQGetSearchList** function takes a list of names and classes and returns a list of database levels where a match might occur.

The **XrmQGetSearchResource** function searches the specified database levels for the resource that is fully identified by the specified name and class.

SEE ALSO

XrmInitialize(3X), **XrmMergeDatabases**(3X), **XrmPutResource**(3X), **XrmUniqueQuark**(3X)
Xlib – C Language X Interface

NAME

XrmInitialize, XrmParseCommand – initialize the Resource Manager and parse the command line

SYNTAX

void **XrmInitialize**(); void **XrmParseCommand**(*database*, *table*, *table_count*, *name*, *argc_return*, *argv_return*,)
XrmDatabase **database*;
XrmOptionDescList *table*;
int *table_count*;
char **name*;
int **argc_return*;
char ***argv_return*;

OPTIONS

argc_return	Contains the number of arguments before the call. After the call, returns the number of remaining arguments.
argv_return	Represents a pointer to the command line arguments before the call. After the call, matched arguments have been removed.
database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.
name	
table	Specifies table of command line arguments to be parsed.
table_count	Specifies the number of entries in the table.

DESCRIPTION

The **XrmInitialize** function initializes the resource manager. The **XrmParseCommand** function parses an (arc, argv) pair according to the specified option table, loads recognized options into the specified database, and modifies the (arc, argv) pair to remove all recognized options.

SEE ALSO

XrmGetResource(3X), **XrmMergeDatabases**(3X), **XrmPutResource**(3X), **XrmUniqueQuark**(3X)
Xlib – C Language X Interface

NAME

XrmMergeDatabases, XrmGetFileDatabase, XrmPutFileDatabase, XrmGetStringDatabase – 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;

OPTIONS

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.

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 **XrmPutFileDatabase** function stores a copy of the application's current database in the specified file.

The **XrmGetStringDatabase** function creates a new database and stores the resources specified in the specified null-terminated string.

SEE ALSO

XrmGetResource(3X), **XrmInitialize**(3X), **XrmPutResource**(3X), **XrmUniqueQuark**(3X)
Xlib – C Language X Interface

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*, *resource*, *value*)
XrmDatabase **database*;
char **resource*;
char **value*;

void **XrmQPutStringResource**(*database*, *bindings*, *quarks*, *value*)
XrmDatabase **database*;
XrmBindingList *bindings*;
XrmQuarkList *quarks*;
char **value*;

void **XrmPutLineResource**(*database*, *line*)
XrmDatabase **database*;
char **line*;

OPTIONS

bindings	Specifies a list of bindings.
database	Specifies a pointer to the resource database. If database contains NULL, a new resource database is created and a pointer to it is returned in database.
line	Specifies the resource value pair as a single string. A single colon (":") separates the name from the value.
quarks	Specifies the partial name or class list of the resource to be stored.
resource	Specifies the resource as a string.
specifier	Specifies a partial specification of the resource.
type	Specifies the type of the resource.
value	Specifies the value of the resource.

DESCRIPTION

The **XrmPutResource** and **XrmQPutResource** functions store a resource specification into the specified database.

XrmPutStringResource adds a resource with the specified value to the specified database.

The **XrmQPutStringResource** function adds a string resource to the specified database using quarks as the specification.

XrmPutLineResource adds a single resource entry to the specified database.

SEE ALSO

XrmGetResource(3X), **XrmInitialize**(3X), **XrmMergeDatabases**(3X), **XrmUniqueQuark**(3X)
Xlib – C Language X Interface

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*;

OPTIONS

bindings_return	Returns the binding list.
quark	Specifies the quark for which the equivalence string is desired.
quark_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.

The **XrmStringToQuark** function converts the specified string to a resource quark representation.

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. The components of the string are separated by a "." character.

The **XrmStringToBindingQuarkList** function converts the specified string to a binding list and a quark list.

SEE ALSO

XrmGetResource(3X), **XrmInitialize**(3X), **XrmMergeDatabases**(3X), **XrmPutResource**(3X)
Xlib – C Language X Interface