68NW9209F46A

Graphic Services Extension Programmer's Reference Manual

MOTOROLA

GRAPHIC SERVICES EXTENSION PROGRAMMER'S REFERENCE MANUAL

Part Number 68NW9209F46A

Version 1

PREFACE

The *Graphics Services Extension Programmer's Reference Manual* (Part Number 68NW9209F46A) describes the C Language programming interface to the X Window System, the X library (Xlib). Note that the Resource Manager manual pages (Xrm*name*(3X)) are grouped after the main body of Xlib manual pages and before the permuted index.

CONTENTS

1. INTRODUCTION

This manual describes the C Language programming interface to the X Window System, the X library (Xlib). This library enables a programmer to write applications with complete network transparency, using an advanced user interface based on windows on the screen.

The X library is the lowest level of programming interface to the X Window System. It is powerful enough to allow you to write effective applications without additional programming tools. The X library is required for certain tasks even in applications written with higher-level "toolkits."

This manual does not deal with toolkits. Nonetheless, all the information described in this book is essential for using toolkits because the toolkits themselves are written using Xlib, and Xlib will be used together with a toolkit in virtually all applications.

The major change between Release 1 and Release 2 of X is the resource manager. The resource manager allows you to easily parse the command line and then merge these preferences with the defaults for the program and the defaults for the user. These operations are standard practice for thoroughly written applications. In Release 1 of Xlib, the resource manager was a separate library, which had been developed as part of the Xtk toolkit. As of Release 2, it has been incorporated into Xlib. This manual describes both the Release 1 and Release 2 resource managers.

Some Assumptions About Experience

This manual assumes that readers are proficient in the C programming language, although examples are provided for infrequently used features of the language that are necessary or useful when programming with X. In addition, general familiarity with the principles of raster graphics is assumed.

How To Use This Manual

The reference pages in Section 1 describe programs intended to be invoked directly by the user. The reference pages in Section 3X describe the calling sequences of all the Xlib functions. The pages describe briefly how the function is normally used when there is a specific, non-obvious technique involved. Note that the resource manager manual pages (Xrm*name*(3X)) are included after the main body of Xlib functions and before the permuted index.

The function entries (3X) in this manual are based on the following format; the command entries (1) follow a similar format. Some entries do not include all the sections listed here or include sections that are specific to the entry (e.g., STARTUP FILE VARIABLES).

NAME
Gives the name of the entry and briefly states its purpose.

SYNOPSIS
Summarizes the use of the program being described.

ARGUMENTS
Explains the nature of the variables and constants passed to the subroutine. The following conventions are observed in the ARGUMENTS section.

- The display argument, where used, is always first in the argument list.
- Resource objects (*Window, Drawable, Font, Pixmap, Cursor, Colormap, GContext,* and *KeySym*), where used, occur at the beginning of the argument list, immediately after the display variable.
- *Drawable*s come before all other resources in the argument list.
- Source arguments always precede the destination arguments in the argument list.
- The *x* argument always precedes the *y* argument, and 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 with a count in the argument list (number of elements in the array), the array always precedes the count.
- Where a structure is accompanied by a mask indicating which members of the structure are to be read, the mask always precedes the pointer to the structure in the argument list.

DESCRIPTION
Details what the function does, what it returns, and what events or side-effects it causes. It also contains miscellaneous information such as examples of usage, special error cases, and references to the *Graphics Services Extension (GSE) Programmer's Guide.*

STRUCTURES
Contains the C definitions of the X-specific data types used by the functions as arguments or return values. It also contains definitions of important constants used by the function.

ERRORS
: Lists the error event types that may be generated by a particular function and identifies the cause of certain errors.

BUGS
: Lists known faults in software that have not been rectified. Occasionally, a suggested short term remedy is also described.

EXAMPLES
: Gives examples of usage, where appropriate.

SEE ALSO
: Gives pointers to related functions in the manual and to related macros covered in the *Graphics Services Extension Programmer's Guide.*

A permuted index is provided at the end of this manual. This is a list of keywords, given in the second of three columns, together with the context in which each keyword is found. The right column lists the name of the manual page on which each keyword may be found. The left column contains useful information about the keyword.

Conventions Used in This Manual

- **Boldface** strings represents pathnames or literals and are to be typed just as they appear.
- *Italic* strings usually represent substitutable argument prototypes, resource objects, or program names found elsewhere in the manual.
- `Constant Width` strings are used for examples of source code.
- Square brackets [ ] around an argument prototype indicate that the argument is optional.

Related Documentation

Graphics Services Extension Programmer's Guide (68NW9209F47A)
: This manual provides tutorial information, examples, and appendices that will be useful for X programmers.

NAME

uwm - a window manager for X

SYNOPSIS

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 variables, **resetbindings, resetmenus,** and **resetvariables** at the top of that specific startup file.

OPTIONS

-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 menu cursor in first menu item. If unspecified, 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 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 to the left and right of 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 color map. **maxcolors** counts colors as they are included in the file.

normali/nonormali
Places icons created with **f.newiconify** within the root window, even if it is placed partially off the screen. With **nonormali** the icon is placed exactly where the cursor leaves it.

normalw/nonormalw
Places 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, and **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.

resizerelative/noresizerelative
Indicates whether or not resize operations should be done relative to moving edge or edges. By default, the dynamic rectangle uses the actual pointer location to define the new size.

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 above and below 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* "

Function and mouse events are required input. 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 will 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 lower case, 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 lower case 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	Function occurs when the specified button is pressed down.
up	Function 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.

Some applications use the mouse buttons in their tasks. If *uwm* has bound any of the mouse buttons without modifier keys, then these will be unavailable to the application. In other words, if a *uwm* function or menu is invoked by just pressing a mouse key without a keyboard modifier, then that mouse key will be unavailable to the application regardless of context.

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

Begin with an exclamation point (!) and 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 (^) will have a new line character appended to the string after the up arrow (^) has been stripped from it.

Strings starting with a bar character (|) will be copied as is after the bar character (|) has been 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 color map 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 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  :w|i     :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

xset(1), *xlsfonts*(1).

COPYRIGHT

AUTHOR

M. Gancarz, DEC Ultrix Engineering Group, Merrimack, New Hampshire, using some algorithms originally by Bob Scheifler, MIT Laboratory for Computer Science.

NAME

xclock - analog / digital clock for X

SYNOPSIS

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 once 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 preferred size and position of the clock window.

–display *host:display*
: Specifies the X server to contact.

–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
Get the default host and display number.

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

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.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHORS

Tony Della Fera (MIT-Athena, DEC)
Dave Mankins (MIT-Athena, BBN)
Ed Moy (UC Berkeley)

NAME

xfd - font displayer for X

SYNOPSIS

xfd [-options ...] [fontname]

OPTIONS

–bw *number*
Allows you to specify the width of the window border in pixels.

–rv The foreground and background colors will be switched. The default colors are black on white.

–fw Overrides a previous choice of reverse video. The foreground and background colors will not be switched.

–fg *color* On color displays, determines the foreground color (the color of the text).

–bg *color*
On color displays, determines the background color.

–bd *color*
On color displays, determines the color of the border.

–bf *fontname*
Specifies the font to be used for the messages at the bottom of the window.

–tl *title* Specifies the name of an *xfd* icon.

–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. This produces a distracting display when used with a background color.

–start *charnum*
Specifies that character number *charnum* should be the first character displayed.

–geometry *geometry*
Specifies an initial window geometry.

–display *display*
Specifies the display to use.

DESCRIPTION

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

The characters are displayed in a grid of boxes, each large enough to hold any character of the font. If the **-gray** option has been supplied, the characters will be displayed using *XDrawImageString* using the foreground and background colors on a gray background. This permits determining exactly how *XDrawImageString* will draw any given character. If **-gray** has not been supplied, the characters will simply be drawn using the foreground color on the background color.

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

Note that if the font is an 8-bit font, the characters 256-511 (0x100-0x1ff), 512-767 (0x200-0x2ff), ... will display exactly the same as the characters 0-255 (0x00-0xff). *xfd* by default creates a window of size sufficient to display the first 256 characters using a 16 by 16 grid. In this case, there is no need to scroll forward or backward whole windows in order to see the entire contents of a 8-bit font. Of course, this window may not fit on the screen.

Clicking the middle button on a character will cause that character's number to be displayed in both decimal and hexadecimal at the bottom of the window. If verbose mode is selected, additional information about that particular character will be displayed as well. The displayed information includes the width of the character, its left bearing, right bearing, ascent, and its descent. If verbose mode is selected, typing '<' or '>' into the window will display the minimum or maximum values, respectively taken on by each of these fields over the entire font.

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

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 until '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
Set the border width of the window.

BorderColor
Set the border color of the window.

ReverseVideo
If "on", reverse the definition of foreground and background color.

Foreground
Set the foreground color.

Background
Set the background color.

BodyFont
Set the font to be used in the body of the window (i.e., for messages, etc.). This is not the font that *xfd* displays, just the font it uses to display information about the font being displayed.

IconName
Set the name of the icon.

IconBitmap
Set the file we should look in to get the bitmap for the icon.

Title Set the title to be used.

SEE ALSO

xlsfonts(1).

ENVIRONMENT

DISPLAY

Get the default host and display to use.

XENVIRONMENT

Get the name of a resource file that overrides the global resources stored in the RESOURCE_MANGER property.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xlsfonts - server font list displayer for X

SYNOPSIS

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:dpy*
Specifies the X server to contact.

–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

xfd(1), *xset*(1).

ENVIRONMENT

DISPLAY
to get the default host and display to use.

BUGS

Invoking **xlsfonts -l** can tie up your server for a very long time. This is a bug with single-threaded non-preemptable servers, not with this program.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xprop - property displayer for X

SYNOPSIS

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 used to display window and font properties in an X server. One window or font is selected using the command line arguments or, 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 the user to select window *id* on the command line rather than using the pointer to select the target window. This is very useful in debugging X applications when the target window is not mapped to the screen or when the use of the pointer 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 pointer to select the target window.

-font *font*
Allows the user 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 when the root window is completely obscured.

-display *display*
Allows the user to specify the server to connect to.

-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*(1), or by name if the window possesses a name. The **-id** argument selects a window by id number in either decimal or hex (must start with 0x) while the **-name** argument selects a window by name.

The last way to select a window does not involve the command line at all. If none of **-font, -id, -name,** and **-root** are specified, a crosshairs cursor is displayed and the user 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.

Except for the above four arguments, the **-help** argument for obtaining help, and the **-grammar** argument for listing the full grammar for the command line, the command line arguments are used in specifying both the format of the properties to be displayed and how to display them. The **-len** *n* argument specifies that at most *n* bytes of any given property will be read and displayed. This is useful for example when displaying the cut buffer on the root window which could run to several pages if displayed in full.

Normally each property name is displayed by printing first the property name then its type (if it has one) in parentheses followed by its value. The **-notype** argument specifies that property types should not be displayed. The **-fs** argument is used to specify a file containing a list of formats for properties while the **-f** argument is used to specify the format for one property.

The formatting information for a property actually consists of two parts, a *format* and a *dformat*. The *format* specifies the actual formatting of the property (i.e., is it made up of words, bytes, or longs?, etc.) while the *dformat* specifies how the property should be displayed.

The following paragraphs describe how to construct *format*s and *dformat*s. However, for the vast majority of users and uses, this should not be necessary as the built in defaults contain the *format*s and *dformat*s necessary to display all the standard properties. It should only be necessary to specify *format*s and *dformat*s if a new property is being dealt with or the user dislikes the standard display format. New users especially are encouraged to skip this part.

A *format* consists of one of 0, 8, 16, or 32 followed by a sequence of one or more format characters. The 0, 8, 16, or 32 specifies how many bits per field there are in the property. Zero is a special case that specifies using 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. A sequence of words is byte swapped while a sequence of bytes is not when read by a machine of the opposite byte order of the machine that originally wrote the property.

Once the size of the fields has been specified, it is necessary to specify the type of each field (i.e., integer, string, atom, etc.) 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 manner similar to the formatting string used by *printf*. For example, the *dformat* " is ($0, $1 \)\n" would render the POINT 3, -4 which has a *format* of 32ii as " is (3, -4)\n".

Any character other than a $, ?, \, or a (in a *dformat* prints as itself. To print one of the following characters: $, ?, \, or (, precede it by a \. For example, to print out a $, use \$. Several special backslash sequences are provided as shortcuts. \n will cause a newline to be displayed while \t will cause a tab to be displayed. *o* where *o* is an octal number will display character number *o*.

A $ followed by a number *n* causes field number *n* to be displayed. The format of the displayed field depends on the formatting character used to describe it in the corresponding *format*. For example, if a cardinal is described by 'c', it will print in decimal; 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 and so on until the last field defined. If field *n* is not defined, nothing is displayed. This is useful for a property that is a list of values.

A ? is used to start a conditional expression, a kind of if-then statement. ?*exp*(*text*) will display *text* if and only if *exp* evaluates to non-zero. This is useful for two reasons. First, it allows fields to be displayed if and only if a flag is set. And second, it allows a value to such as a state number to be displayed as a name rather than as just a number. The syntax of *exp* is as follows:

exp	::= *term* | *term*=*exp* | !*exp*
term	::= *n* | $*n* | m*n*

The ! operator is a logical "not", changing 0 to 1 and any non-zero value to 0. = is an equality operator. Note that internally all expressions are evaluated as 32-bit numbers, so -1 is not equal to 65535. = returns 1 if the two values are equal and 0 if not. *n* represents the constant value *n* while $*n* represents the value of field number *n*. m*n* is 1 if flag number *n* in the first field having format character 'm' in the corresponding *format* is 1, 0 otherwise.

Examples: ?m3(count: $3\n) displays field 3 with a label of count if and only if flag number 3 (count starts at 0!) is on. ?$2=0(True)?!$2=0(False) displays the inverted value of field 2 as a boolean.

In order to display a property, *xprop* needs both a *format* and a *dformat*. Before *xprop* uses its default values of a *format* of 32x and a *dformat* of " = { $0+ }\n", it searches several places in an attempt to find more specific formats. First, a search is made using the name of the property. If this fails, a search is made using the type of the property. This allows type STRING to be defined with one set of formats while allowing property WM_NAME which is of type STRING to be defined with a different format. In this way, the display formats for a given type can be overridden for specific properties.

The locations searched are in order: the format if any specified with the property name (as in 8x WM_NAME), the formats defined by **-f** options in last to first order, the contents of the file specified by the **-fs** option if any, the contents of the file specified by the environmental variable XPROP-FORMATS if any, and finally *xprop*'s built in file of formats.

The format of the files referred to by the **-fs** argument and the XPROP-FORMATS 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 point size of the fixed font: *xprop* -font fixed POINT_SIZE

To display all the properties of window # 0x200007: *xprop* -id 0x200007

ENVIRONMENT

DISPLAY
Get default display.

XPROPFORMATS
Specify the name of a file from which additional formats are to be obtained.

SEE ALSO

xwininfo(1).

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xrefresh - refresh all or part of an X screen

SYNOPSIS

xrefresh [-option ...]

DESCRIPTION

xrefresh is a simple X program that causes all or part of the 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 Use a white background. The screen just appears to flash quickly and then repaint.

–black Use a black background (in effect, turning off all of the electron guns to the tube). This can be somewhat disorienting as everything goes black for a moment.

–solid *color*
Use a solid background of the specified color. Try green.

–root Use the root window background.

–none This is the default. All of the windows simply repaint.

–geometry *WxH+X+Y*
Specifies the portion of the screen to be repainted. This supercedes the old style **=WxH+X+Y**.

–display *display*
This argument allows you to specify the server and screen to refresh.

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.

BUGS

It should have just one default type for the background.

COPYRIGHT

AUTHORS

Jim Gettys, Digital Equipment Corp., MIT Project Athena

NAME

xset - user preference utility for X

SYNOPSIS

xset [-display *display*] [-b] [b on/off] [b [*volume* [*pitch* [*duration*]]] [-c] [c on/off] [c [*volume*]] [fp *path*[,*path*[,...]]] [fp default] [[-]led [*integer*]] [led on/off] [m[ouse] [*acceleration* [*threshold*]]] [m[ouse] default] [p *pixel color*] [[-]r] [r on/off] [s [*length* [*period*]]] [s blank/noblank] [s expose/noexpose] [s on/off] [s default] [q]

DESCRIPTION

This program is used to set various user preference options of the display.

OPTIONS

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

b
: Controls 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 key click. This option can take an optional value, a preceding dash(-), or an 'on/off' flag. If no parameter or the 'on' flag is given, the system defaults will be used. If the dash or 'off' flag is used, keyclick will be disabled. If a value from 0 to 100 is given, it is used to indicate volume as a percentage of the maximum. The X server will set the volume to the nearest value that the hardware can support.

fp
: Sets the font path. It must be followed by a comma-separated list of directories or the flag 'default'. The indicated path will be 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 will be turned on or off depending on the existence of a preceding dash. A common LED which can be controlled is the "Caps Lock" LED. "xset led 3" would turn led #3 on. "xset -led 3" would turn it off. The particular LED values may refer to different LEDs on different hardware.

m 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 with a flick of the wrist when desired. One or both parameters for the **m** option can be omitted, but if only one is given, it will be interpreted as the acceleration. If no parameters or the flag 'default' is used, the system defaults will be set.

p Controls pixel color values. The parameters are the color map entry number in decimal and a color specification. The root background colors may be changed on some servers by altering the entries for BlackPixel and WhitePixel. Although these are often 0 and 1, they need not be. Also, a server may choose to allocate those colors privately, in which case an error will be generated. The map entry must not be a read-only color, or an error will result.

r Controls the autorepeat. If a preceding dash or the 'off' flag is used, autorepeat will be disabled. If no parameters or the 'on' flag is used, autorepeat will be enabled.

s Allows the user 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 will be set to its default screen saver characteristics. The 'on/off' flags simply turn the screen saver functions on or off. The 'blank' flag sets the preference to blank the video (if the hardware can do so) rather than display a background pattern, while 'noblank' sets the preference to display a pattern rather than blank the video. The 'expose' flag sets the preference to allow window exposures (the server can freely discard window contents), while 'noexpose' sets the preference to disable screen saver unless the server can regenerate the screens without causing exposure events. The length and period parameters for the screen saver function determines how long the server must be inactive for screen saving to activate, and the period to change the background pattern to avoid burn in. The arguments are specified in seconds. If only one numerical parameter is given, it will be used for the length.

q Gives information on the current settings.

These settings will be reset to default values when you log out.

Note that not all X implementations are guaranteed to honor all of these options.

SEE ALSO

xsetroot(1).

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Bob Scheifler, MIT Laboratory for Computer Science
David Krikorian, MIT Project Athena (X11 version)

NAME

xsetroot – root window parameter setting utility for X

SYNOPSIS

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

DESCRIPTION

The *setroot* program allows you to tailor the appearance of the background ("root") window on a workstation display running X. Normally, you experiment with *xsetroot* until you find a background 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 change options (**-solid, -gray, -grey, -bitmap,** and **-mod**) may be specified at a time.

OPTIONS

The various options are as follows:

-help Prints a usage message and exit.

-def Resets unspecified attributes to the default values. (Restores the background to the familiar gray mesh and the cursor to the hollow x shape.)

-cursor *cursorfile maskfile*
Allows you to change the pointer cursor to whatever you want when the pointer cursor is outside of any window. Cursor and mask files are bitmaps (little pictures). You may 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. 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 ALSO

xset(1).

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

xterm – terminal emulator for X

SYNOPSIS

xterm [*-toolkitoption* ...] [-option ...]

DESCRIPTION

The *xterm* program is a terminal emulator for the Graphics Services Extension. It provides DEC VT102-compatible terminals for programs that cannot use the window system directly. If the underlying operating system supports terminal resizing capabilities (for example, the SIGWINCH signal in systems derived from 4.3bsd), *xterm* will use the facilities to notify programs running in the window whenever it is resized.

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

–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* program to display lines that are exactly the width of the window and are followed by line beginning with a tab to be displayed incorrectly (the leading tabs are not displayed).

+cu Indicates that 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 not do jump scrolling.

+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 make *xterm* much faster when scanning through large amounts of text. The VT100 escape sequences for enabling and disabling smooth scroll as well as the "Modes" menu can be used to turn this feature on or off.

–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 the shell that is started in the *xterm* window be a login shell (i.e. the first character of argv[0] will be a dash, indicating to the shell that it should read the user's .login or .profile).

+ls Indicates that the shell that is 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 margin bell should not be rung.

–ms *color*
Specifies the color to be used for the pointer 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 position 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 VT102 mode.

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

+vb 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 by users when starting terminal windows.**

–*Sccn* 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 preferred position of the icon window. It is shorthand for specifying the "**iconGeometry*" resource.

–T *string*
Specifies the title for *xterm*'s icons. 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 preferred size and position of the VT102 window.

–display *display*
: Specifies the X server to contact.

–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 preferred size and position of the application when iconified. It is not necessarily obeyed by all window managers.

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

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 preferred size and position of the VT102 window.

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

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

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 "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 entries that work with *xterm* include "xterm", "vt102", "vt100" and "ansi", and *xterm* automatically searches the termcap file in this order for these entries and then sets the "TERM" and the "TERMCAP" environment variables.

Many of the special *xterm* features (like logging) may be modified under program control through a set of escape sequences different from the standard VT102 escape sequences.

POINTER USAGE

Once the VT102 window is created, *xterm* allows you to select text and copy it within the same or other windows.

The selection functions are invoked when the pointer buttons are used with no modifiers, and when they are used with the "shift" key.

Pointer button one (left) 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 is saved in the global cut buffer when the button is released. Double-clicking selects by words. Triple-clicking selects by lines. Quadruple-clicking goes back to characters, etc. Multiple-click is determined by the time from button up to button down, so you can change the selection unit in the middle of a selection.

Pointer button two (middle) 'types' (pastes) the text from the cut buffer, inserting it as keyboard input.

Pointer button three (right) extends the current selection. ("right" and "left" are interchangeable 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 treat the cut buffer as if it were a text file, i.e., the text is delimited by new lines.

The scroll region displays the position and amount of text currently showing in the window (highlighted) relative to the amount of text actually saved. As more text is saved (up to the maximum), the size of the highlighted area decreases.

Clicking button one with the pointer in the scroll region moves the adjacent line to the top of the display window.

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

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

MENUS

xterm has two different menus, named **xterm** and **Modes.** 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. At the bottom portion of the menu are command entries; selecting one of these performs the indicated function.

The **xterm** menu pops up when the "control" key and pointer button one are pressed in a window. Notable entries in the command section of the menu are the **Continue, Suspend, Interrupt, Hangup, Terminate** and **Kill** which send 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.

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* entry for *xterm* allows the visual editor *vi*(1) to switch to the alternate screen for editing, and restore the screen on exit.

There are escape sequences in VT102 mode to change the name of the windows and to specify a new log file name.

ENVIRONMENT

xterm sets the environment variables "TERM" and "TERMCAP" properly for the size window you have created. It also uses and sets the environment variable "DISPLAY" to specify which bit map display terminal to use. The environment variable "WINDOWID" is set to the X window id number of the *xterm* window.

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.

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.

NOTE

If any of the keys used within *xterm* are bound by the window manager (*uwm*) without keyboard modifiers (e.g., Alt, Shift, Cntl), the key functions are unavailable to *xterm*.

COPYRIGHT

Copyright 1988, Massachusetts Institute of Technology.

AUTHORS

Loretta Guarino Reid (DEC-UEG-WSL), Joel McCormack (DEC-UEG-WSL), Terry Weissman (DEC-UEG-WSL), Edward Moy (Berkeley), Ralph R. Swick (MIT-Athena), Mark Vandevoorde (MIT-Athena), Bob McNamara (DEC-MAD), Jim Gettys (MIT-Athena), Bob Scheifler (MIT X Consortium), Doug Mink (SAO), Steve Pitschke (Stellar), Ron Newman (MIT-Athena), Jim Fulton (MIT X Consortium), et al.

NAME

xwininfo - window information utility for X

SYNOPSIS

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 when the target window is not mapped to the screen or when 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 root, parent, and children windows' ids and the names of the selected window to be displayed.

-stats Causes various attributes of the selected window having to do with its location and appearance 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 its 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 number and name is, where the window's icon should go, and what the window's initial state should be.

-all A quick way to ask for all information possible.

-display *display*
Allow the user to specify the server to connect to.

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

Get default host and display number.

SEE ALSO

xprop(1).

COPYRIGHT

AUTHOR

Mark Lillibridge, MIT Project Athena

NAME

XActivateScreenSaver — activate screen blanking.

SYNOPSIS

```
XActivateScreenSaver (display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XActivateScreenSaver turns on the screen saver using the parameters set with *XSetScreenSaver*. This means that the screen may go blank, or some random change to the display will take place to save the phosphors from burnout.

SEE ALSO

XForceScreenSaver, XResetScreenSaver, XGetScreenSaver, XSetScreenSaver.

NAME

XAddHost — add a host to the access control list.

SYNOPSIS

```
XAddHost(display, host)
  Display *display;
  XHostAddress *host;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
host	Specifies the network address of the host machine to be added.

DESCRIPTION

XAddHost adds the specified host to the access control list for the specified display. The display hardware associated with the program that issues this command must be on the host whose list is being updated. The access control list is a primitive security feature.

The *address* data must be a valid address for the type of network in which the server operates, as specified in the *family* member.

STRUCTURES

```
typedef struct {
  int family;                 /* for example AF_DNET */
  int length;                 /* length of address, in bytes */
  char *address;              /* pointer to where to find the bytes
} XHostAddress;

/* The following constants for family member */
#define FamilyInternet        0
#define FamilyDECnet          1
#define FamilyChaos           2
```

ERRORS

BadAlloc
BadValue

SEE ALSO

XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XDisableAccessControl, XEnableAccessControl, XSetAccessControl.

NAME

XAddHosts — add multiple hosts to the access control list.

SYNOPSIS

```
XAddHosts (display, hosts, num_hosts)
  Display  *display;
  XHostAddress  *hosts;
  int num_hosts;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
hosts	Specifies each host that is to be added.
num_hosts	Specifies the number of hosts that is to be added.

DESCRIPTION

XAddHosts adds each specified host to the access control list for the display. The display hardware associated with the program that issues this command must be on the host whose list is being updated. The access control list is a primitive security feature.

The *address* data must be a valid address for the type of network in which the server operates, as specified by the *family* member.

STRUCTURES

```
typedef struct {
  int family;                   /* for example AF_DNET */
  int length;                   /* length of address, in bytes */
  char *address;                /* pointer to where to find the bytes */
} XHostAddress;

/* The following constants for family member */
#define FamilyInternet          0
#define FamilyDECnet            1
#define FamilyChaos             2
```

ERRORS

BadAlloc
BadValue

SEE ALSO

XAddHost, XListHosts, XRemoveHost, XRemoveHosts, XDisableAccessControl, XEnableAccessControl, XSetAccessControl, XAddHost, XListHosts, XRemoveHost, XRemoveHosts, XDisableAccessControl, XEnableAccessControl, XSetAccessControl.

NAME

XAddPixel — add constant value to every pixel value in image.

SYNOPSIS

```
int XAddPixel(ximage, value)
  XImage *ximage;
  int value;
```

ARGUMENTS

ximage	Specifies a pointer to the image.
value	Specifies the constant value that is to be added. Valid pixel value ranges depend on the visual used to create the image. If this value added to the existing value overflows, extra bits in the result are truncated.

DESCRIPTION

XAddPixel adds a constant value to every pixel value in an image. This function is useful when you have a base pixel value derived from the allocation of color resources and need to manipulate an image so that the pixel values are in the same range.

STRUCTURES

```
typedef struct _XImage {
   int width, height;        /* size of image */
   int xoffset;              /* number of pixels offset in X direction */
   int format;               /* XYBitmap, XYPixmap, ZPixmap */
   char *data;               /* pointer to image data */
   int byte_order;           /* data byte order, LSBFirst, MSBFirst */
   int bitmap_unit;          /* quant. of scanline 8, 16, 32 */
   int bitmap_bit_order;     /* LSBFirst, MSBFirst */
   int bitmap_pad;           /* 8, 16, 32 either XY or ZPixmap */
   int depth;                /* depth of image */
   int bytes_per_line;       /* accelarator to next line */
   int bits_per_pixel;       /* bits per pixel (ZPixmap) */
   unsigned long red_mask;   /* bits in z arrangment */
   unsigned long green_mask;
   unsigned long blue_mask;
   char *obdata;             /* hook for the object routines to hang on */
   struct funcs {            /* image manipulation routines */
   struct _XImage *(*create_image)();
   int (*destroy_image)();
   unsigned long (*get_pixel)();
```

```
            int (*put_pixel)();
            struct _XImage *(*sub_image)();
            int (*add_pixel)();
            } f;
        } XImage;
```

SEE ALSO

XDestroyImage, XPutImage, XGetImage, XCreateImage, XSubImage, XGetSubImage, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XAddToSaveSet — add window's children to client's save-set.

SYNOPSIS

```
XAddToSaveSet(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the window whose children you want to add to the client's save-set.

DESCRIPTION

XAddToSaveSet adds the children of the specified window to client's save-set.

The save-set is a safety net for windows that have been reparented by the window manager, usually to provide a shadow or other background for each window. When the window manager dies unexpectedly, the windows in the save-set are reparented to their closest living ancestor, so that they remain alive. Refer to the *GSE Programmer's Guide* for more information about save-sets.

Use *XRemoveFromSaveSet* to remove a window's children from the client's save-set.

ERRORS

BadMatch	*w* not created by some other client.
BadWindow	

SEE ALSO

XRemoveFromSaveSet, XChangeSaveSet.

NAME

XAllocColor — allocate a read-only colormap cell with closest hardware-supported color.

SYNOPSIS

```
Status XAllocColor(display, cmap, colorcell_def)
  Display *display;
  Colormap cmap;
  XColor *colorcell_def;        /* reads and RETURNs */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap ID.
colorcell_def	Specifies desired RGB values, and also returns the pixel value and the RGB values actually used in the colormap.

DESCRIPTION

XAllocColor returns in the *XColor* structure the pixel value of a read-only (shareable) colorcell with the closest RGB values available in *cmap*. *XAllocColor* also returns the red, green, and blue values actually used.

If the display hardware has an immutable hardware colormap, the entire colormap will be read-only, and the closest cell that exists will be returned. Otherwise, the colormap is read/write, and may have some read/write cells, some read-only cells, and some unallocated. If a read-only cell exists that matches the requested RGB values, that cell is returned. If no matching cell exists but there are unallocated cells, a cell is allocated to match the specified RGB values. If no matching cell exists and there are no unallocated cells, the closest available colorcell that has already been allocated (by this or any other client) is returned. Note that *colorcell_def* stores both the requested color when *XAllocColor* is called and the result when *XAllocColor* returns.

XAllocColor returns 0 if there is a problem (typically all cells are allocated and read/write), or 1 if it succeeds.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                    /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadAlloc

BadColor

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XAllocColorCells — allocate read/write (non-shared) colorcells.

SYNOPSIS

```
Status XAllocColorCells(display, cmap, contig, plane_masks,
nplanes, pixels, ncolors)
  Display *display;
  Colormap cmap;
  Bool contig;
  unsigned long plane_masks[nplanes];/* RETURN */
  unsigned int nplanes;
  unsigned long pixels[ncolors]; /* RETURN pixel values*/
  unsigned int ncolors;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
cmap	Specifies the colormap.
contig	Specifies a boolean value. Pass *True* if the planes must be contiguous or *False* if the planes need not be contiguous.
plane_mask	Returns an array of plane masks.
nplanes	Specifies the number of plane masks returned in the plane masks array. Must be non-negative.
pixels	Returns an array of pixel values.
ncolors	Specifies the number of pixel values returned in the pixels array. Must be positive.

DESCRIPTION

XAllocColCells allocates read/write colorcells in a read/write colormap. If *ncolors* and *nplanes* are requested, then *ncolors* pixels and *nplanes* plane masks are returned. No mask will have any bits in common with any other mask, or with any of the pixels. By ORing together each of the *pixels* with any combination of the *plane_masks, ncolors* * $2^{nplanes}$ distinct pixels can be produced. For *GrayScale* or *PseudoColor,* each mask will have exactly one bit, and for *DirectColor* each will have exactly three bits. If *contig* is *True,* then if all plane masks are ORed together, a single contiguous set of bits will be formed for *GrayScale* or *PseudoColor* and three contiguous sets of bits (one within each pixel subfield) for *DirectColor.* The RGB values of the allocated entries are undefined until set with *XStoreColor* or like functions.

Status is 0 on failure.

ERRORS

BadAlloc	
BadColor	
BadValue	*nplanes* not non-negative. *ncolors* not positive.

SEE ALSO

XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XAllocColorPlanes — allocate read/write (non-sharable) color planes.

SYNOPSIS

```
Status XAllocColorPlanes(display, cmap, contig, pixels,
ncolors, nreds, ngreens, nblues, rmask, gmask, bmask)
  Display *display;
  Colormap cmap;
  Bool contig;
  unsigned long pixels[ncolors];          /* RETURN */
  int ncolors;
  int nreds, ngreens, nblues;
  unsigned long *rmask, *gmask, *bmask;   /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap ID.
contig	Specifies a boolean value. Pass *True* if the planes must be contiguous or *False* if the planes do not need to be contiguous.
pixels	Returns an array of pixel values.
ncolors	Specifies the number of pixel values returned in the pixels array. Must be positive.
nreds *ngreens* *nblues*	Specify the number of red, green, and blue colors (shades). Must be non-negative.
rmask *gmask* *bmask*	Return bit masks for the red, green, and blue planes.

DESCRIPTION

If *ncolors, nreds, ngreens,* and *nblues* are requested, then *ncolors pixels* are returned, and the masks have *nreds, ngreens,* and *nblues* bits set respectively. If *contig* is *True,* then each mask will have a contiguous set of bits. No mask will have any bits in common with any other mask, or with any of the pixels. For *DirectColor,* each mask will lie within the corresponding pixel subfield. By ORing together subsets of masks with pixels, *ncolors**($2^{(nreds+ngreens+nblues)}$) distinct pixels can be produced. All of these are allocated by the request. However, in the colormap there are only *ncolors**(2^{nreds}) independent red entries, *ncolors**($2^{ngreens}$) independent green entries, and *ncolors**(2^{nblues}) independent blue entries. This is true even for *PseudoColor.* When the colormap entry for a pixel value is changed using *XStoreColors* or *XStoreNamedColor,* the pixel is decomposed according to the masks and the corresponding pixel subfield entries are updated.

Status is 0 on failure.

ERRORS

BadAlloc	
BadColor	
BadValue	*ncolors* not positive. *nreds, ngreens, nblues* not non-negative.

SEE ALSO

XAllocColorCells, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XAllocNamedColor — allocate read-only colorcell from color name.

SYNOPSIS

```
Status XAllocNamedColor(display, cmap, colorname,
colorcell_def, rgb_db_def)
  Display *display;
  Colormap cmap;
  char *colorname;
  XColor *colorcell_def;        /* RETURN */
  XColor *rgb_db_def;           /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
colorname	Specifies the color name string (for example, "red") you want. ISO Latin-1 encoding, upper/lower case does not matter.
colorcell_def	Returns the pixel value and RGB values actually used in the colormap. This is the closest color supported by the hardware.
rgb_db_def	Returns the exact RGB values from the database corresponding to the *colorname* supplied.

DESCRIPTION

XAllocNamedColor determines the RGB values for the specified *colorname* from the color database, and then allocates a read-only color cell with the closest color available, as described under *XAllocColor*. Both the 'exact' data base definition of the color, and the color actually allocated are returned. If the colormap is not full, the RGB values allocated are the closest supported by the hardware. If the colormap is full, it returns the closest read-only colorcell already allocated, and does not actually create or set any new colorcell.

XAllocNamedColor returns a Status of 0 when it encounters an error or 1 when it succeeds.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                  /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadAlloc
BadColor
BadName

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XAllowEvents — control the behavior of keyboard and pointer events when these resources are grabbed.

SYNOPSIS

```
XAllowEvents(display, event_mode, time)
  Display *display;
  int event_mode;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event_mode	Specifies the event mode. Pass one of these constants: *AsyncPointer, SyncPointer, AsyncKeyboard, SyncKeyboard, ReplayPointer, ReplayKeyboard, AsyncBoth,* or *SyncBoth*.
time	Specifies the time when the grab should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime*.

DESCRIPTION

XAllowEvents releases the events queued in the server since the last *XAllowEvents* call for the same device and by the same client. Events are queued in the server only when the client has caused a device to "freeze" (by grabbing the device with mode *GrabModeSync*). The request has no effect if *time* is earlier than the last-grab time or later than the current server time.

The *event_mode* argument controls what device events are released for and just how and when they are released. The *event_mode* is interpreted as follows:

AsyncPointer If *XAllowEvents* is called with *AsyncPointer* while the pointer is frozen by the client, pointer event processing resumes normally, even if the pointer is frozen twice by the client on behalf of two separate grabs. *AsyncPointer* has no effect if the pointer is not frozen by the client, but the pointer need not be grabbed by the client.

AsyncKeyboard If *XAllowEvents* is called with *AsyncKeyboard* while the keyboard is frozen by the client, the keyboard event processing resumes normally, even if the keyboard is frozen twice by the client on behalf of two separate grabs. *AsyncKeyboard* has no effect if the keyboard is not frozen by the client, but the keyboard need not be grabbed by the client.

SyncPointer If *XAllowEvents* is called with *SyncPointer* while the pointer is frozen by the client, normal pointer event processing continues until the next *ButtonPress* or *ButtonRelease* event is reported to the client. At this time, the pointer again appears to freeze. However, if the reported event causes the pointer grab to be released, then the pointer does not freeze, which is the case when an automatic grab is released by a *ButtonRelease* or when *XGrabButton* or *XGrabKey* has been called and the specified key or button is released. *SyncPointer* has no effect if the pointer is not frozen or not grabbed by the client.

SyncKeyboard If *XAllowEvents* is called with *SyncKeyboard* while the keyboard is frozen by the client, normal keyboard event processing continues until the next *KeyPress* or *KeyRelease* event is reported to the client. At this time, the keyboard again appears to freeze. However, if the reported event causes the keyboard grab to be released, then the keyboard does not freeze, which is the case when an automatic grab is released by a *ButtonRelease* or when *XGrabButton* or *XGrabKey* has been called and the specified key or button is released. *SyncKeyboard* has no effect if the keyboard is not frozen or not grabbed by the client.

ReplayPointer This symbol has an effect only if the pointer is grabbed by the client and thereby frozen as the result of an event. In other words, *XGrabButton* must have been called and the selected button/key combination pressed, or an automatic grab (initiated by a *ButtonPress*) must be in effect, or a previous *XAllowEvents* must have been called with mode *SyncPointer.* If the *pointer_mode* of the *XGrabPointer* was *GrabModeSync,* then the grab is released and the releasing event is processed as if it had occured after the release, ignoring any passive grabs at or above in the hierarchy (towards the root) on the grab-window of the grab just released.

ReplayKeyboard This symbol has an effect only if the keyboard is grabbed by the client and if the keyboard is frozen as the result of an event. In other words, *XGrabKey* must have been called and the selected key combination pressed, or a previous *XAllowEvents* must have been called with mode *SyncKeyboard*. If the *pointer_mode* or *keyboard_mode* of the *XGrabKey* was *GrabModeSync*, then the grab is released and the releasing event is processed as if it had occured after the release, ignoring any passive grabs at or above in the hierarchy (towards the root)

SyncBoth *SyncBoth* has the effect described for both *SyncKeyboard* and *SyncPointer*. *SyncBoth* has no effect unless both pointer and keyboard are frozen by the client. If the pointer or keyboard is frozen twice by the client on behalf of two separate grabs, *SyncBoth* "thaws" for both (but a subsequent freeze for *SyncBoth* will only freeze each device once).

AsyncBoth *AsyncBoth* has the effect described for both *AsyncKeyboard* and *AsyncPointer*. *AsyncBoth* has no effect unless both pointer and keyboard are frozen by the client. If the pointer and the keyboard were frozen by the client, or if both are frozen twice by two separate grabs, event processing (for both devices) continues normally. If a device is frozen twice by the client on behalf of the two separate grabs, *AsyncBoth* releases events for both.

AsyncPointer, *SyncPointer*, and *ReplayPointer* have no effect on the processing of keyboard events. *AsyncKeyboard*, *SyncKeyboard*, and *ReplayKeyboard* have no effect on the processing of pointer events.

It is possible for both a pointer grab and a keyboard grab (by the same or different clients) to be active simultaneously. If a device is frozen on behalf of either grab, no event processing is performed for the device. It is also possible for a single device to be frozen because of both grabs. In this case, the freeze must be released on behalf of both grabs before events can again be processed.

ERRORS

BadValue	Invalid mode constant.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XAutoRepeatOff — turn off the keyboard auto-repeat keys.

SYNOPSIS

```
XAutoRepeatOff(display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XAutoRepeatOff turns off auto-repeat for the keyboard. It sets the keyboard so that holding a key down will not result in multiple events.

SEE ALSO

XGetDefault, XAutoRepeatOn, XBell, XGetKeyboardControl, XChangeKeyboardControl, XGetPointerControl.

NAME

XAutoRepeatOn — turn on the keyboard auto-repeat keys.

SYNOPSIS

```
XAutoRepeatOn(display)
  Display *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XAutoRepeatOn sets the keyboard to auto-repeat; that is, holding a key down will result in multiple *KeyPress* and *KeyRelease* event pairs with the same *keycode* member.

SEE ALSO

XGetDefault, XAutoRepeatOff, XBell, XGetKeyboardControl, XChangeKeyboardControl, XGetPointerControl.

NAME

XBell — ring the bell (Control G).

SYNOPSIS

```
XBell(display, percent)
  Display *display;
  int percent;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
percent	Specifies the volume for the bell, relative to the base volume set with *XChangeKeyboardControl*. Possible values are –100 (off), through 0 (base volume), to 100 (loudest) inclusive.

DESCRIPTION

Rings the bell on the keyboard at a volume relative to the base volume for the keyboard, if possible. *percent* can range from –100 to 100 inclusive (else a *BadValue* error). The volume at which the bell is rung when *percent* is non-negative is:

```
volume = base - [(base * percent) / 100] + percent
```

and when *percent* is negative:

```
volume = base + [(base * percent) / 100]
```

To change the base volume of the bell, set the *bell_percent* variable of *XChangeKeyboardControl*.

ERRORS

BadValue	*percent* < –100 or *percent* >100.

SEE ALSO

XGetDefault, XAutoRepeatOff, XAutoRepeatOn, XGetKeyboardControl, XChangeKeyboardControl, XGetPointerControl.

NAME

XChangeActivePointerGrab — change parameters of active pointer grab.

SYNOPSIS

```
XChangeActivePointerGrab(display, event_mask, cursor, time)
                Display *display;
                unsigned int event_mask;
                Cursor cursor;
                Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event_mask	Specifies which pointer events are reported to the client. This mask is the bitwise OR of one or more of these pointer event masks: *ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2-Motion-Mask, Button5-Motion-Mask, Button-Motion-Mask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask, ButtonMotionMask, KeyMapStateMask.*
cursor	Specifies the cursor that is displayed. A possible value you can pass is *None,* which will keep the current cursor.
time	Specifies the time when the grab should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime.*

DESCRIPTION

XChangeActivePointerGrab changes the specified dynamic parameters if the pointer is actively grabbed by the client and the specified time is no earlier than the last pointer grab time and no later than the current X server time. *XChangeActivePointerGrab* has no effect on the passive parameters of *XGrabButton,* or the automatic grab that occurs between *ButtonPress* and *ButtonRelease.*

event_mask is always augmented to include *ButtonPress* and *ButtonRelease.*

ERRORS

BadCursor

SEE ALSO

XQueryPointer, XWarpPointer, XGrabPointer, XUngrabPointer, XGetPointerMapping, XSetPointerMapping, XGetPointerControl, XChangePointerControl.

NAME

XChangeGC — change components of a graphics context.

SYNOPSIS

```
XChangeGC(display, gc, valuemask, values)
  Display *display;
  GC gc;
  unsigned long valuemask;
  XGCValues *values;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
valuemask	Specifies the components in the graphics context that you want to change. This argument is the bitwise OR of one or more of the GC component masks.
values	Specifies a pointer to the **XGCValues** structure.

DESCRIPTION

XChangeGC changes any or all of the components of a GC. The *valuemask* specifies which components are to be changed. The *values* structure contains the values to be set. These two arguments operate just like they do in *XCreateGC*. Changing the *clip_mask* overrides any previous *XSetClipRectangles* request for this GC. Changing the *dash_offset* or *dash_list* overrides any previous *XSetDashes* request on this GC.

Since consecutive changes to the same GC are buffered, there is no advantage to using this routine over the routines that set individual members of the GC.

Even if an error occurs, a subset of the components may have already been altered.

STRUCTURES

```
typedef struct {
  int function;                 /* logical operation */
  unsigned long plane_mask; /* plane mask */
  unsigned long foreground; /* foreground pixel */
  unsigned long background; /* background pixel */
  int line_width;               /* line width */
  int line_style;               /* LineSolid,LineOnOffDash,LineDoubleDash */
  int cap_style;                /* CapNotLast,CapButt,CapRound,CapProjecting */
  int join_style;            /* JoinMiter, JoinRound, JoinBevel */
  int fill_style;            /* FillSolid, FillTiled, FillStippled */
  int fill_rule;             /* EvenOddRule, WindingRule */
  int arc_mode;              /* ArcChord, ArcPieSlice */
  Pixmap tile;               /* tile pixmap for tiling operations */
  Pixmap stipple;            /* stipple 1 plane pixmap for stipping */
  int ts_x_origin;           /* offset for tile or stipple operations */
  int ts_y_origin;
  Font font;                 /* default text font for text operations */
  int subwindow_mode;        /* ClipByChildren, IncludeInferiors */
  Bool graphics_exposures;   /* generate events on XCopy, Area, XCopyPlane*/
  int clip_x_origin;         /* origin for clipping */
  int clip_y_origin;
  Pixmap clip_mask;          /* bitmap clipping; other calls for rects */
  int dash_offset;           /* patterned/dashed line information */
  char dashes;
} XGCValues;

     #define GCFunction              (1L<<0)
     #define GCPlaneMask             (1L<<1)
     #define GCForeground            (1L<<2)
     #define GCBackground            (1L<<3)
     #define GCLineWidth             (1L<<4)
     #define GCLineStyle             (1L<<5)
     #define GCCapStyle              (1L<<6)
     #define GCJoinStyle             (1L<<7)
     #define GCFillStyle             (1L<<8)
     #define GCFillRule              (1L<<9)
     #define GCTile                  (1L<<10)
     #define GCStipple               (1L<<11)
     #define GCTileStipXOrigin       (1L<<12)
     #define GCTileStipYOrigin       (1L<<13)
     #define GCFont                  (1L<<14)
     #define GCSubwindowMode         (1L<<15)
     #define GCGraphicsExposures     (1L<<16)
     #define GCClipXOrigin           (1L<<17)
     #define GCClipYOrigin           (1L<<18)
     #define GCClipMask              (1L<<19)
     #define GCDashOffset            (1L<<20)
     #define GCDashList              (1L<<21)
     #define GCArcMode               (1L<<22)
```

ERRORS

BadAlloc
BadFont
BadGC
BadMatch
BadPixmap
BadValue

SEE ALSO

XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetRegion, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XChangeKeyboardControl — change keyboard preferences such as key click.

SYNOPSIS

```
XChangeKeyboardControl(display, value_mask, values)
  Display *display;
  unsigned long value_mask;
  XKeyboardControl *values;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
value_mask	Specifies a mask composed of ORed symbols from the table at the end of this page, specifying which fields to set.
values	Specifies the settings for the keyboard preferences.

DESCRIPTION

XChangeKeyboardControl sets user preferences such as key click, bell volume and duration, LED state, and keyboard autorepeat.

The *value_mask* argument specifies which values are to be changed; the *values* structure contains the values to be set.

key_click_percent sets the volume for key clicks between 0 (off) and 100 (loud) inclusive, if possible. Setting to –1 restores the default.

The *bell_percent* sets the base volume for the bell between 0 (off) and 100 (loud) inclusive, if possible. Setting to –1 restores the default. The *bell_pitch* sets the pitch (specified in Hz) of the bell, if possible. Setting to –1 restores the default. The *bell_duration* sets the duration (specified in milliseconds) of the bell, if possible. Setting to -1 restores the default. A bell generator connected with the console but not directly on a keyboard is treated as if it were part of the main keyboard.

If both *led_mode* and *led* are specified, then the state of that LED is changed, if possible. If only *led_mode* is specified, then the state of all LEDs are changed, if possible. At most 32 LEDs are supported, numbered starting from 1.

If both *auto_repeat_mode* and *key* are specified, then the *auto_repeat* mode of that key is changed, if possible. If only *auto_repeat_mode* is specified, then the global *auto_repeat* mode for the entire keyboard is changed, if possible, without affecting the *per_key* settings.

The order in which the changes are performed is server dependent, and some may be completed when another causes an error.

STRUCTURES

```
/* masks for ChangeKeyboardControl */

#define KBKeyClickPercent       (1L<<0)
#define KBBellPercent           (1L<<1)
#define KBBellPitch             (1L<<2)
#define KBBellDuration          (1L<<3)
#define KBLed                   (1L<<4)
#define KBLedMode               (1L<<5)
#define KBKey                   (1L<<6)
#define KBAutoRepeatMode        (1L<<7)

/* structure for ChangeKeyboardControl */

typedef struct {
  int key_click_percent;
  int bell_percent;
  int bell_pitch;
  int bell_duration;
  int led;
  int led_mode
  int key;
  int auto_repeat_mode;         /* AutoRepeatModeOff, AutoRepeatModeOn,
                                AutoRepeatModeDefault */
} XKeyboardControl;
```

ERRORS

BadMatch — *values.key* specified but *values.auto.repeat.mode* not specified.
values.led specified but *values.led_mode* not specified.

BadValue — *values.key_click_percent* < –1.
values.bell_percent < –1.
values.bell_pitch < –1.
values.bell_duration <-1.

SEE ALSO

XGetDefault, XAutoRepeatOff, XAutoRepeatOn, XBell, XGetKeyboardControl, XGetPointerControl.

NAME

XChangeKeyboardMapping — change keyboard mapping.

SYNOPSIS

XChangeKeyboardMapping(*display*, *first_code*, *keysyms_per_code*, *keysyms*, *num_codes*)
  Display **display*;
  int *first_keycode*;
  int *keysyms_per_keycode*;
  KeySym **keysyms*;
  int *num_keycodes*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
first_keycode	Specifies the first keycode that is to be changed.
keysyms_per_keycode	Specifies the number of keysyms that the caller is supplying for each keycode.
keysyms	Specifies a pointer to the list of *KeySyms*.
num_keycodes	Specifies the number of keycodes that are to be changed.

DESCRIPTION

Starting with *first_keycode, XChangeKeyboardMapping* defines the symbols for the specified number of keycodes. The symbols for keycodes outside this range remained unchanged. The number of elements in the keysyms list must be a multiple of *keysyms_per_keycode* (else a *BadLength* error). The specified *first_keycode* must be greater than or equal to *min_keycode* supplied at connection setup and stored in the display structure (else a *BadValue* error). In addition, the following expression must be less than or equal to *max_keycode* as returned in the connection setup (else a *BadValue* error).

```
max_keycode >= first_keycode + (num_keycodes / keysyms_per_keycode) - 1
```

The *KeySym* number N (counting from zero) for keycode K has an index (counting from zero) of the following (in keysyms).

```
index = (K - first_keycode) * keysyms_per_keycode + N
```

The specified *keysyms_per_keycode* can be chosen arbitrarily by the client to be large enough to hold all desired symbols. A special *KeySym* value of *NoSymbol* should be used to fill in unused elements for individual keycodes. It is legal for *NoSymbol* to appear in nontrailing positions of the effective list for a keycode.

XChangeKeyboardMapping generates a *MappingNotify* event.

ERRORS

BadAlloc	
BadLength	Number of elements in *keysyms* not multiple of *keysyms_per_keycode*.
BadValue	*first.keycode* less than *display->min_keycode*. *display->max_keycode* exceeded (see above).

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XChangePointerControl — change pointer acceleration.

SYNOPSIS

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*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
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 threshold is set. You can pass one of these constants: *True* or *False*.
accel_numerator	Specifies the numerator for the acceleration multiplier.
accel_denominator	Specifies the denominator for the acceleration multiplier.
threshold	Specifies the acceleration threshold.

DESCRIPTION

XChangePointerControl defines how the pointing device moves. The acceleration is a fraction (*accel_numerator/accel_denominator*) which specifies how many times faster than normal the pointer moves compared to how fast it normally moves. Acceleration takes affect only when a particular pointer motion is greater than *threshold* pixels at once, and only applies to the motion beyond *threshold* pixels. The values for *do_accel* and *do_threshold* must be non-zero for the pointer values to be set; otherwise, the parameters will be unchanged. Setting any argument to -1 restores the default for that argument.

The fraction may be rounded arbitrarily by the server.

ERRORS

BadValue	*accel_denominator* is zero. Negative value for *do_accel* or *do_threshold*.

SEE ALSO

XQueryPointer, XWarpPointer, XGrabPointer, XChangeActivePointerGrab, XUngrabPointer, XGetPointerMapping, XSetPointerMapping, XGetPointerControl.

NAME

XChangeProperty — change a property associated with a window.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the window whose property you want to change.
property	Specifies the property atom.
type	Specifies the type of the property. X does not interpret the type, but simply passes it back to an application that later calls *XGetProperty*.
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*.
mode	Specifies the mode of the operation. Possible values are *PropModeReplace, PropModePrepend, PropModeAppend,* or no value.
data	Specifies the property data.
nelements	Specifies the number of elements in the property.

DESCRIPTION

XChangeProperty changes a property and generates *PropertyNotify* events if they have been selected.

XChangeProperty does the following according to the *mode* argument:

- *PropModeReplace* – Discards the previous property value.

- *PropModePrepend* – Inserts the data before the beginning of the existing data. If the property is undefined, it is treated as defined with the correct type and format with zero length data. *type* and *format* arguments must match the existing property value, otherwise a *BadMatch* error occurs.

- *PropModeAppend* – Appends the data onto the end of the existing data. If the property is undefined, it is treated as defined with the correct type and format with zero length data. *type* and *format* arguments must match the existing property value, otherwise a *BadMatch* error occurs.

The property may remain defined even after the client which defined it exits. The property becomes undefined only if the application calls *XDeleteProperty,* destroys the specified window, or closes the last connection to the X server.

The maximum size of a property is server dependent and can vary dynamically if the server has sufficient memory.

ERRORS

BadAlloc
BadAtom
BadMatch
BadValue
BadWindow

SEE ALSO

XSetStandardProperties, XGetFontProperty, XRotateWindowProperties, XDeleteProperty, XGetWindowProperty, XListProperties, XGetAtomName, XInternAtom.

NAME

XChangeSaveSet — add or remove a subwindow from the client's save-set.

SYNOPSIS

```
XChangeSaveSet(display, w, change_mode)
  Display *display;
  Window w;
  int change_mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose children you want to add or remove from the client's save-set; it must have been created by some other client.
change_mode	Specifies the mode. Pass one of these constants: *SetModeInsert* (adds the window to this client's save-set) or *SetModeDelete* (deletes the window from this client's save-set).

DESCRIPTION

XChangeSaveSet controls the longevity of subwindows, which are normally destroyed when the parent is destroyed.

The save-set of a client is a list of other client's windows which, if they are inferiors of one of the client's windows at connection close, should not be destroyed and should be remapped if they are unmapped. For example, a window manager which wants to add decoration to a window by adding a "frame," might reparent an application's window to the "frame window." When the frame is destroyed, the application's window should not also be destroyed, but should be returned to its previous place in the window hierarchy. Refer to the *GSE Programmer's Guide* for more information about save-sets.

Windows are removed automatically from the save-set by the server when they are destroyed. For each window in the client's save-set, if the window is an inferior of a window created by the client, the save-set window is reparented to the closest ancestor such that the save-set window is not an inferior of a window created by the client. If the save-set window is unmapped, a *MapWindow* request is performed on it. After save-set processing, all windows created by the client are destroyed. For each non-window resource created by the client, the appropriate Free request is performed. All colors and colormap entries allocated by the client are freed.

ERRORS

BadMatch	*w* not created by some other client.
BadValue	
BadWindow	

SEE ALSO

XAddToSaveSet, XRemoveFromSaveSet.

NAME

XChangeWindowAttributes — set window attributes.

SYNOPSIS

```
XChangeWindowAttributes(display, w, valuemask, attributes)
  Display *display;
  Window w;
  unsigned long valuemask;
  XSetWindowAttributes *attributes;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
valuemask	Specifies which window attributes are defined in the *attributes* argument. If *valuemask* is 0, the rest is ignored, and *attributes* is not referenced. The values and restrictions are the same as for *XCreateSimpleWindow* and *XCreateWindow*.
attributes	Window attributes to be changed. The *valuemask* indicates which members in this structure are referenced.

DESCRIPTION

XChangeWindowAttributes changes any or all of the window attributes that can be changed. For descriptions of the window attributes, refer to the *GSE Programmer's Guide*.

Changing the background does not cause the window contents to be changed. Use *XClearWindow* to cause the background to be repainted. Setting the border, or changing the background such that the border tile origin changes, causes the border to be repainted. Changing the background of a root window to *None* or *ParentRelative* restores the default background pixmap. Changing the border of a root window to *CopyFromParent* restores the default border pixmap.

Changing the *win_gravity* does not affect the current position of the window. Changing the *backing_store* of an obscured window to *WhenMapped* or *Always* may have no immediate effect. Also changing the *backing_planes, backing_pixel,* or *save_under* of a mapped window may have no immediate effect.

Multiple clients can select input on the same window; the *event_mask* passed are disjoint. When an event is generated it will be reported to all interested clients. Therefore, the setting of the *event_mask* attribute by one client will not affect the *event_mask* of others on the same window. However, at most, one client at a time can select each of *SubstructureRedirectMask, ReSizeRedirectMask,* and *ButtonPressMask* on any one window. If a client attempts to select on *SubtructureRedirectMask, ResizeRedirectMask,* or *ButtonPressMask* and some other client has already selected it on the same window, the X server generates a *BadAccess* error.

There is only one *do_not_propagate_mask* for a window, not one per client.

Changing the colormap attribute of a window generates a *ColormapNotify* event. Changing the colormap attribute of a visible window may have no immediate effect on the screen (because the map may not be installed until the window manager or client calls *XInstallColormap*).

Changing the cursor of a root window to *None* restores the default cursor.

STRUCTURES

```
/*
 * Data structure for setting window attributes.
 */
typedef struct {
  Pixmap background_pixmap;      /*backgrnd pixmap,None,orParentRelative*/
  unsigned long background_pixel;/* background pixel */
  Pixmap border_pixmap;         /* border of the window */
  unsigned long border_pixel;   /* border pixel value */
  int bit_gravity;              /* one of bit gravity values */
  int win_gravity;              /* one of the window gravity values */
  int backing_store;            /* NotUseful, WhenMapped, Always */
  unsigned long backing_planes;/* planes to be preseved if possible */
  unsigned long backing_pixel; /* value to use in restoring planes */
  Bool save_under;              /* should bits under be saved (popups) */
  long event_mask;              /* set of events that should be saved */
  long do_not_propagate_mask;   /* set of events that should not propagate */
  Bool override_redirect;       /* override redirected config request */
  Colormap colormap;            /* colormap to be associated with window */
  Cursor cursor;                /* cursor to be displayed (or None) */
} XSetWindowAttributes;
```

```
/* Window attributes for CreateWindow and ChangeWindowAttributes */

/* Definitions for valuemask argument */

#define CWBackPixmap            (1L<<0)
#define CWBackPixel             (1L<<1)
#define CWBorderPixmap          (1L<<2)
#define CWBorderPixel           (1L<<3)
#define CWBitGravity            (1L<<4)
#define CWWinGravity            (1L<<5)
#define CWBackingStore          (1L<<6)
#define CWBackingPlanes         (1L<<7)
#define CWBackingPixel          (1L<<8)
#define CWOverrideRedirect      (1L<<9)
#define CWSaveUnder             (1L<<10)
#define CWEventMask             (1L<<11)
#define CWDontPropagate         (1L<<12)
#define CWColormap              (1L<<13)
#define CWCursor                (1L<<14)
```

ERRORS

BadAccess
BadColor
BadCursor
BadMatch
BadPixmap
BadValue
BadWindow

SEE ALSO

XGetWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap, XGetGeometry.

NAME

XCheckIfEvent — check event queue for matching event.

SYNOPSIS

```
int XCheckIfEvent(display, event, predicate, args)
  Display *display;
  XEvent *event;              /* RETURN */
  Bool (*predicate)();
  char *args;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event	Returns the matched event.
predicate	Specifies the procedure that is called to determine if the next event matches your criteria.
args	Specifies the user-specified arguments that will be passed to the predicate procedure.

DESCRIPTION

XCheckIfEvent returns the next event in the queue that is matched by the specified predicate procedure. That event is removed from the queue. If no match is found, *XCheckIfEvent* returns *False* and flushes the output buffer. No other events are removed from the queue. Later events in the queue are not searched.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XCheckMaskEvent — remove next event that matches mask but do not wait.

SYNOPSIS

```
Bool XCheckMaskEvent(display, mask_event, event)
  Display *display;
  long mask_event;
  XEvent *event;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event_mask	Specifies the event types to be returned. See list under *XSelectInput*.
event	Returns a copy of the matched event's *XEvent* structure.

DESCRIPTION

XCheckMaskEvent removes the next event in the queue which matches the passed mask. The event is copied into an *XEvent* supplied by the caller and *XCheckMaskEvent* returns *True*. Other events earlier in the queue are not discarded. If no such event has been queued, *XCheckMaskEvent* flushes the output buffer and immediately returns *False*, without waiting.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XCheckTypedEvent — return next event in queue that matches event type but do not wait.

SYNOPSIS

```
Bool XCheckTypedEvent(display, event_type, report)
  Display *display;
  int event_type;
  XEvent *report;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event_type	Specifies the event type to be compared.
report	Returns a copy of the matched event structure.

DESCRIPTION

XCheckTypedEvent searches first the event queue, then the events available on the server connection, for the specified *event_type*. If there is a match, it returns the associated event structure. Events searched but not matched are not discarded. *XCheckTypedEvent* returns *True* if the event is found. If the event is not found, *XCheckTypedEvent* flushes the output buffer and returns *False*.

This command is similar to *XCheckMaskEvent*, but it searches through the queue instead of inspecting only the last item on the queue. It also matches only a single event type instead of multiple event types as specified by a mask.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XCheckTypedWindowEvent — return next event in queue matching type and window.

SYNOPSIS

```
Bool XCheckTypedWindowEvent(display, w, event_type, report)
  Display *display;
  Window w;
  int event_type;
  XEvent *report;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
event_type	Specifies the event type to be compared.
report	Returns the matched event's associated structure into this client-supplied structure.

DESCRIPTION

XCheckTypedWindowEvent searches first the event queue, then any events available on the server connection for an event that matches the specified window and the specified event type. Events searched but not matched are not discarded.

XCheckTypedWindowEvent returns *True* if the event is found; it flushes the output buffer and returns *False* if the event is not found.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XCheckWindowEvent — remove next event matching both passed window and passed mask, but do not wait.

SYNOPSIS

```
Bool XCheckWindowEvent(display, w, event_mask, event)
  Display *display;
  Window w;
  long event_mask;
  XEvent *event;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. The event must match both the passed window and the passed event mask.
event_mask	Specifies the event mask. See *XSelectInput* for a list of mask elements.
event	Returns the *XEvent* structure.

DESCRIPTION

XCheckWindowEvent removes the next event in the queue which matches both the passed window and the passed mask. If such an event exists, it is copied into an *XEvent* supplied by the caller. Other events earlier in the queue are not discarded.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

RETURNED VALUE

If a matching event is found, *XCheckWindowEvent* returns *True*. If no such event has been queued, it flushes the output buffer and returns *False*, without waiting.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XCirculateSubwindows — circulate stacking order of children up or down order.

SYNOPSIS

```
XCirculateSubwindows(display, w, direction)
  Display *display;
  Window w;
  int direction;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the parent of the subwindows to be circulated.
direction	Specifies the direction (up or down) that you want to circulate the children. Pass either *RaiseLowest* or *LowerHighest*.

DESCRIPTION

XCirculateSubwindows circulates the children of the specified window in the specified direction, either *RaiseLowest* or *LowerHighest*. If some other client has selected *SubstructureRedirectMask* on the specified window, then a *CirculateRequest* event is generated, and no further processing is performed. If you specify **RaiseLowest**, this function raises the lowest mapped child (if any) that is occluded by another child to the top of the stack. If you specify **LowerHighest**, this function lowers the highest mapped child (if any) that occludes another child to the bottom of the stack. Exposure processing is performed on formerly obscured windows.

ERRORS

BadValue
BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XCirculateSubwindowsDown — circulate bottom child to top of stacking order.

SYNOPSIS

```
XCirculateSubwindowsDown(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the parent of the windows to be circulated.

DESCRIPTION

XCirculateSubwindowsDown lowers the highest mapped child of the specified window that partially or completely obscures another child. The lowered child goes to the bottom of the stack. Completely unobscured children are not affected. Generates exposure events on any window formerly obscured. Repeated executions lead to round-robin lowering. This is equivalent to *XCirculateSubwindows* (*display, w, LowerHighest*).

If some other client has selected *SubstructureRedirectMask* on the window, then a *CirculateRequest* event is generated, and no further processing is performed.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XCirculateSubwindowsUp — circulate top child to bottom of stacking order.

SYNOPSIS

```
XCirculateSubwindowsUp(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the parent of the windows to be circulated.

DESCRIPTION

XCirculateSubwindowsUp raises the lowest mapped child of the specified window that is partially or completely obscured by another child. The raised child goes to the top of the stack. Completely unobscured children are not affected. This generates exposure events on the raised child (and its descendents, if any). Repeated executions lead to round robin-raising. This is equivalent to *XCirculateSubwindows* (*display, w, RaiseLowest*).

If some other client has selected *SubstructureRedirectMask* on the window, then a *CirculateRequest* event is generated, and no further processing is performed.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XClearArea — clear a rectangular area in a window.

SYNOPSIS

```
XClearArea(display, w, x, y, width, height, exposures)
  Display *display;
  Window w;
  int x, y;
  unsigned int width, height;
  Bool exposures;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of an *InputOutput* window.
x *y*	Specify the x and y coordinates. These coordinates are relative to the origin of the window and specify the upper left corner of the rectangle.
width *height*	Specify the dimensions of the rectangle to be cleared.
exposures	Specifies whether exposure events are generated. Must be either *True* or *False*.

DESCRIPTION

XClearArea clears a rectangular area in a window.

If *width* is zero, the window is cleared from *x* to the right edge of the window. If *height* is zero, the window is cleared from *y* to the bottom of the window.

If the window has a defined background tile or it is *ParentRelative,* the rectangle is tiled with a *plane_mask* of all ones and *function* of *GXCopy.* If the window has background *None,* the contents of the window are not changed. In either case, if *exposures* is *True,* then one or more exposure events are generated for regions of the rectangle that are either visible or are being retained in a backing store.

ERRORS

BadMatch	Window is an *InputOnly* class window.
BadValue	
BadWindow	

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearWindow.

NAME

XClearWindow — clear an entire window.

SYNOPSIS

```
XClearWindow(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XClearWindow clears a window, but does not cause exposure events. This function is equivalent to *XClearArea(display, w,* `0, 0, 0, 0, False)`.

If the window has a defined background tile or it is **`ParentRelative`**, the rectangle is tiled with a *plane_mask* of all ones and *function* of *GXCopy*. If the window has background *None*, the contents of the window are not changed.

ERRORS

BadMatch	If *w* is an *InputOnly* class window.
BadValue	
BadWindow	

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea.

NAME

XClipBox — generate smallest rectangle enclosing region.

SYNOPSIS

```
XClipBox(r, rect)
  Region r;
  XRectangle *rect;          /*  RETURN  */
```

ARGUMENTS

r	Specifies the region.
rect	Returns the smallest rectangle enclosing region.

DESCRIPTION

XClipBox returns the smallest rectangle that encloses the given region.

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
  unsigned short width, height;
} XRectangle;
/*
 * opaque reference to Region data type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion.

NAME

XCloseDisplay — disconnect client from an X server and display.

SYNOPSIS

XCloseDisplay(*display*)
  Display **display*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XCloseDisplay closes the connection between the current client and the X server specified by the *Display* argument.

The *XCloseDisplay* routine destroys all windows, resource IDs (*Window, Font, Pixmap, Colormap, Cursor,* and *GContext*), or other resources (GCs) that the client application has created on this display, unless the *CloseDownMode* of the client's resources has been changed by *XSetCloseDownMode.* Therefore, these windows, resource IDs, and other resources should not be referenced again. In addition, this routine discards any output that has been buffered but not yet sent. Although these operations automatically (implicitly) occur when a process exits, you should call *XCloseDisplay* anyway.

SEE ALSO

XFree, XOpenDisplay, XNoOp,

NAME

XConfigureWindow — change window position, size, border width, or stacking order.

SYNOPSIS

```
XConfigureWindow(display, w, value_mask, values)
  Display *display;
  Window w;
  unsigned int value_mask;
  XWindowChanges *values;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of window to be reconfigured.
value_mask	Specifies which values are to be set using information in the *values* structure. *value_mask* is the bitwise inclusive OR of the valid change window value bits. See their definitions in the Structures section below.
values	Specifies a pointer to the *XWindowChanges* structure containing new configuration information. See the Structures section below.

DESCRIPTION

XConfigureWindow changes the window position, size, border width, and/or the stacking order. This call should not be made without preparing for interaction with the window manager. A *ConfigureNotify* event is generated to announce any changes.

If the *override_redirect* attribute of the window is *False,* and if some other client has selected *SubstructureRedirectMask* on the parent, then the X server generates a *ConfigureRequest* event, and no further processing is performed. If some other client has selected *ResizeRedirectMask* on the window and *width* or *height* is being changed, then a *ResizeRequest* event is generated and the actual size of the window is not changed. The *ResizeRequest* event will be received by the other client (the window manager) and some action taken. The client should wait for the *ConfigureNotify* event to find out the size of the window. Note that the *override_redirect* attribute of the window has no effect on *ResizeRedirectMask* and that *SubstructureRedirectMask* on the parent has precedence over *ResizeRedirectMask* on the window.

When the geometry of the window is changed as specified, the window is restacked among siblings, and a *ConfigureNotify* event is generated if the state of the window actually changes. X generates *GravityNotify* events after generating *ConfigureNotify* events.

If a window's size actually changes, the window's subwindows may move according to their window gravity. Depending on the window's bit gravity, the contents of the window also may be moved. Refer to the *GSE Programmer's Guide* for further information.

Exposure processing is performed on formerly obscured windows, including the window itself and its inferiors, if regions of them were obscured but now are not. As a result of increasing the width or height, exposure processing is also performed on any new regions of the window and any regions where window contents are lost.

The members of *XWindowChanges* that you specify in *values* are:

x *y*	Specify the x and y coordinates relative to the parent's origin and the position of the upper left outer corner of the window.
width *height*	Specify the inside size of the window, not including the border. These arguments must be positive.
border_width	Specifies the width of the border in pixels.
sibling	Specifies the sibling window for stacking operations. If not specified, no change in the stacking order will be made. If specified, *stack_mode* must also be specified.
stack_mode	The stack mode can be any of these constants: *Above, Below, TopIf, BottomIf,* or *Opposite.*

The computation for the *BottomIf, TopIf,* and *Opposite* stacking modes is performed with respect to the *w*'s final size and position (as controlled by the other arguments to *XConfigureWindow,* not its initial position.) It is an error if *sibling* is specified without *stack_mode.* If *sibling* and *stack_mode* are specified, the window is restacked as follows:

Stacking Flag	Position
Above	*w* is placed just above *sibling*
Below	*w* is placed just below *sibling*
TopIf	if *sibling* obscures *w,* then *w*

	is placed at the top of the stack
BottomIf	if *w* obscures *sibling*, then *w* is placed at the bottom of the stack
Opposite	if *sibling* occludes *w*, then *w* is placed at the top of the stack, else if *w* occludes *sibling*, then *w* is placed at the bottom of the stack

If a *stack_mode* is specified but no sibling is specified, the window is restacked as follows:

Stacking Flag	Position
Above	*w* is placed at the top of the stack
Below	*w* is placed at the bottom of the stack
TopIf	if any sibling obscures *w*, then *w* is placed at the top of the stack
BottomIf	if *w* obscures any sibling, then window is placed at the bottom of the stack
Opposite	if any sibling occludes *w*, then *w* is placed at the top of the stack, else if *w* occludes any sibling, then *w* is placed at the bottom of the stack

STRUCTURES

```
typedef struct {
  int x, y;
  int width, height;
  int border_width;
  Window sibling;
  int stack_mode;
} XWindowChanges;

/* ConfigureWindow structure */
/* ChangeWindow value bits definitions for valuemask */
#define CWX                     (1<<0)
#define CWY                     (1<<1)
#define CWWidth                 (1<<2)
#define CWHeight                (1<<3)
#define CWBorderWidth           (1<<4)
#define CWSibling               (1<<5)
#define CWStackMode             (1<<6)
```

ERRORS

BadMatch	Non-zero *border-width* of *InputOnly* window. *sibling* specified without a *stack_mode*. The *sibling* window is not actually a sibling.
BadValue	*width* or *height* is zero.
BadWindow	

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XQueryTree.

NAME

XConvertSelection — use the value of a selection.

SYNOPSIS

```
XConvertSelection(display, selection, target, property, requestor,
time)
  Display *display;
  Atom selection, target;
  Atom property;              /* may be None */
  Window requestor;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
selection	Specifies the selection atom. *XA_PRIMARY* and *XA_SECONDARY* are the standard selection atoms.
target	Specifies the atom of the target type property.
property	Specifies a property describing the requested data. *None* is also valid.
requestor	Specifies the requesting window.
time	Specifies the time when the conversion should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime*.

DESCRIPTION

XConvertSelection causes a *SelectionRequest* event to be sent to the current selection owner if there is one, specifying the property to store the data in (*selection*), the format to convert that data into before storing it (*target*), the specific information requested (*property*), the window that wants the information (*requestor*), and the time to make the conversion (*time*).

The selection owner responds by sending a *SelectionNotify* event, which confirms the selected atom and type. If no owner for the specified selection exists, or if the owner could not convert to the type specified by requestor, the X server generates a *SelectionNotify* event to the *requestor* with property *None*. Refer to the *GSE Programmer' s Guide* for a description of selection events and selection conventions.

ERRORS

BadAtom
BadWindow

SEE ALSO

XSetSelectionOwner, *XGetSelectionOwner*.

NAME

XCopyArea — copy an area of a drawable.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
src *dest*	Specify the source and destination rectangles to be combined. *src* and *dest* must have the same root and depth.
gc	Specifies the graphics context.
src_x *src_y*	Specify 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*	Specify the dimensions of both the source and destination rectangles.
dest_x *dest_y*	Specify the x and y coordinates within the destination window.

DESCRIPTION

XCopyArea combines the specified rectangle of *src* with the specified rectangle of *dest*. *src* and *dest* must have the same root and depth.

If regions of the source rectangle are obscured and have not been retained in *backing_store,* or if regions outside the boundaries of the source drawable are specified, then those regions are not copied. Instead, the following occurs on all corresponding destination regions that are either visible or are retained in *backing_store.* If *dest* is a window with a background other than *None,* the corresponding regions of the destination are tiled (with *plane_mask* of all ones and function *XGCopy*) with that background. Regardless of tiling, if the destination is a window and *graphics_exposure* in *gc* is *True*, then *GraphicsExpose* events for all corresponding destination regions are generated. If *graphics_exposure* is *True* but no regions are exposed, then a *NoExpose* event is generated.

If regions of the source rectangle are not obscured and *graphics_exposure* is *False*, one *NoExpose* event is generated on the destination.

XCopyArea uses these graphics context components: function, *plane_mask*, *subwindow_mode*, *graphics_exposures*, *clip_x_origin*, *clip_y_origin*, and *clip_mask*.

ERRORS

BadMatch	The *src* and *dest* rectangles do not have the same root and depth.
BadDrawable	
BadGC	

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XCopyColormapAndFree — copy a colormap and return new colormap ID.

SYNOPSIS

```
Colormap XCopyColormapAndFree(display, cmap)
  Display *display;
  Colormap cmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap you are moving out of.

DESCRIPTION

XCopyColormapAndFree is used to obtain a new virtual color map 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 moves all of the client's existing allocations from *cmap* to the returned *Colormap* and frees those entries in *cmap*. Values in other entries of the new *colormap* are undefined. The visual type and screen for the new colormap is the same as for the old.

If *cmap* was created by the client with the *alloc* argument set to *AllocAll*, the new colormap is also created with *AllocAll*, all color values for all entries are copied from *cmap*, and then all entries in *cmap* are freed.

If *cmap* was created with *AllocNone*, the allocations to be moved are all those pixels and planes that have been allocated by the client using *XAllocColor*, *XAllocNamedColor*, *XAllocColorCells*, or *XAllocColorPlanes* and which have not been freed since they were allocated.

ERRORS

BadAlloc
BadColor

SEE ALSO

XCreateColormap, XFreeColormap, XGetStandardColormap, XInstallColormap, XUninstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XCopyGC — copy a graphics context.

SYNOPSIS

```
XCopyGC(display, src, valuemask, dest)
  Display *display;
  GC src, dest;
  unsigned long valuemask;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
src	Specifies the components of the source graphics context.
valuemask	Specifies the components in the source GC structure to be copied into the destination GC.
dest	Specifies the destination graphics context.

DESCRIPTION

XCopyGC copies the selected elements of one Graphics Context to another. Refer to the *GSE Programmer's Guide* for a description of the graphics context.

STRUCTURES

The GC structure contains the following elements:

```
/*
 * Data structure for setting graphics context.
 */
typedef struct {
  int function;                 /* logical operation */
  unsigned long plane_mask;     /* plane mask */
  unsigned long foreground;     /* foreground pixel */
  unsigned long background;     /* background pixel */
  int line_width;               /* line width */
  int line_style;               /* Solid, OnOffDash, DoubleDash */
  int cap_style;                /* NotLast, Butt, Round, Projecting */
  int join_style;               /* Miter, Round, Bevel */
  int fill_style;               /* Solid, Tiled, Stippled */
  int fill_rule;                /* EvenOdd, Winding */
  int arc_mode;                 /* PieSlice */
  Pixmap tile;                  /* tile pixmap for tiling operations */
  Pixmap stipple;               /* stipple 1 plane pixmap for stipping
  int ts_x_origin;              /* offset for tile or stipple operations */
  int ts_y_origin;
  Font font;                    /* default text font for text operations */
  int subwindow_mode;           /* ClipByChildren, IncludeInferiors */
  Bool graphics_exposures;      /* boolean, should exposures be generated */
  int clip_x_origin;            /* origin for clipping */
  int clip_y_origin;
  Pixmap clip_mask;             /* bitmap clipping; other calls for rects */
  int dash_offset;              /* patterned/dashed line information */
  char dashes;
} XGCValues;

/* GC components: masks used in XCreateGC, XCopyGC, XChangeGC, OR'ed into
   GC.stateChanges */


#define GCFunction              (1L<<0)
#define GCPlaneMask             (1L<<1)
#define GCForeground            (1L<<2)
#define GCBackground            (1L<<3)
#define GCLineWidth             (1L<<4)
#define GCLineStyle             (1L<<5)
#define GCCapStyle              (1L<<6)
#define GCJoinStyle             (1L<<7)
#define GCFillStyle             (1L<<8)
#define GCFillRule              (1L<<9)
#define GCTile                  (1L<<10)
#define GCStipple               (1L<<11)
#define GCTileStipXOrigin       (1L<<12)
#define GCTileStipYOrigin       (1L<<13)
#define GCFont                  (1L<<14)
#define GCSubwindowMode         (1L<<15)
#define GCGraphicsExposures     (1L<<16)
#define GCClipXOrigin           (1L<<17)
#define GCClipYOrigin           (1L<<18)
#define GCClipMask              (1L<<19)
#define GCDashOffset            (1L<<20)
#define GCDashList              (1L<<21)
#define GCArcMode               (1L<<22)
```

ERRORS

BadMatch	*src* and *dest* do not have the same root and depth.
BadAlloc	
BadGC	
BadValue	

SEE ALSO

XChangeGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XCopyPlane — copy and color bit-plane of drawable.

SYNOPSIS

```
XCopyPlane(display, src, dest, gc, src_x, src_y, width, height,
dest_x, dest_y, plane)
  Display  *display;
  Drawable src, dest;
  GC gc;
  int src_x, src_y;
  unsigned  int width, height;
  int dest_x, dest_y;
  unsigned  long plane;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
src *dest*	Specify the source and destination drawables.
gc	Specifies the graphics context.
src_x *src_y*	Specify 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*	Specify the width and height. These are the dimensions of both the source and destination rectangles.
dest_x *dest_y*	Specify the x and y coordinates of the copied area relative to the origin of the destination drawable.
plane	Specifies the source bit-plane. You must set exactly one bit.

DESCRIPTION

XCopyPlane copies a single plane of a rectangle in the source into the entire depth of a corresponding rectangle in the destination. The plane of the source drawable and the *foreground/background* pixel values in *gc* are combined to form a pixmap of the same depth as the destination drawable, and the equivalent of an *XCopyArea* is performed, with all the same exposure semantics.

XCopyPlane uses these graphics context components: *function, plane_mask, foreground, background, subwindow_mode, graphics_exposures, clip_x_origin, clip_y_origin,* and *clip_mask.*

src and *dest* must have the same root, but need not have the same depth.

ERRORS

BadDrawable	
BadGC	
BadMatch	*src* and *dest* do not have the same root.
BadValue	*plane* does not have exactly one bit set.

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XCreateAssocTable — create a new association table (X10).

SYNOPSIS

```
XAssocTable  *XCreateAssocTable(size)
  int size;
```

ARGUMENTS

size	Specifies the number of buckets in the hashed association table.

DESCRIPTION

XCreateAssocTable creates an association table, which allows you to associate your own structures with X resources in a fast lookup table. This function is provided for compatibility with X Version 10. To use it you must include the file *<X11/X10.h>* and link with the library *-loldX.*

The *size* argument specifies the number of buckets in the hash system of *XAssocTable.* For reasons of efficiency the number of buckets should be a power of two. Some size suggestions might be: use 32 buckets per 100 objects; a reasonable maximum number of object per buckets is 8. If there is an error allocating memory for the *XAssocTable,* a *NULL* pointer is returned.

STRUCTURES

```
typedef struct {
  XAssoc *buckets;              /* pointer to first bucket in array */
  int size;                     /* table size (number of buckets) */
}XAssocTable;
```

SEE ALSO

XDeleteAssoc, XDestroyAssocTable, XLookUpAssoc, XMakeAssoc.

NAME

XCreateBitmapFromData — create bitmap from X11 bitmap format data.

SYNOPSIS

```
Pixmap XCreateBitmapFromData(display, d, data, width,
height)
  Display *display;
  Drawable d;
  char *data;
  unsigned int width, height;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable. This determines which screen to create the bitmap on.
data	Specifies the location of the bitmap data.
width *height*	Specify the dimensions of the created bitmap. If smaller than the original bitmap, the upper left corner is used.

DESCRIPTION

XCreateBitmapFromData creates a single plane pixmap from an array of hexadecimal data. This data may be defined in the program or included. The bitmap data must be in X version 11 format. *XCreateBitmapFromData* creates an image with the specified data and copies it into the created pixmap. The following format is assumed for the data:

```
format=XYPixmap
bit_order=LSBFirst
byte_order=LSBFirst
bitmap_unit=8
bitmap_pad=8
xoffset=0
no extra bytes per line
```

The following is an example of creating a bitmap:

```
#define gray_width 16
#define gray_height 16
#define gray_x_hot 8
#define gray_y_hot 8
```

```
static char gray_bits[] =
  {
  0xf81f, 0xe3c7, 0xcff3, 0x9ff9,
  0xbffd, 0x33cc, 0x7ffe, 0x7ffe,
  0x7e7e, 0x7ffe, 0x37ec, 0xbbdd,
  0x9c39, 0xcff3, 0xe3c7, 0xf81f
  };

Pixmap XCreateBitmapFromData(display, window, gray_bits, gray_width, gray_height);
```

If insufficient working storage was allocated, *XCreateBitmapFromData* returns *NULL*. The user should free the bitmap using *XFreePixmap* when it is no longer needed.

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreatePixmapFromBitmapData.

NAME

XCreateColormap — create a colormap.

SYNOPSIS

```
Colormap XCreateColormap(display, w, visual, alloc)
  Display *display;
  Window w;
  Visual *visual;
  int alloc;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies a window ID. The colormap created will be associated with the same screen as the window.
visual	Specifies a pointer to the `Visual` structure for the colormap. The visual class and depth must be supported by the screen.
alloc	Specifies how many colormap entries to allocate. Pass either *AllocNone* or *AllocAll.*

DESCRIPTION

XCreateColormap creates a colormap of the specified visual type and allocates either none or all of its entries, and returns the colormap ID.

It is legal to specify any visual class in the structure pointed to by the *visual* argument. If the class is *StaticColor, StaticGray,* or *TrueColor,* the colorcells will have pre-allocated read-only values defined by the individual server but unspecified by the X11 protocol. In these cases, *alloc* must be specified as *AllocNone* (else a *BadMatch* error).

For the other visual classes, *PseudoColor, DirectColor,* and *GrayScale,* you can pass either *AllocAll* or *AllocNone* to the *alloc* argument. If you pass *AllocNone,* the colormap has no allocated entries. This allows your client programs to allocate read-only colorcells with *XAllocColor* or read/write cells with *XAllocColorCells, AllocColorPlanes* and *XStoreColors.* If you pass the constant *AllocAll,* the entire colormap is allocated writable (all the entries are read/write, non-shareable and have undefined initial values), and the colors can be set with *XStoreColors.* However, you cannot free these entries with *XFreeColors,* and no relationships between the entries are defined.

If the visual class is *PseudoColor* or *GrayScale* and *alloc* is *AllocAll,* this function simulates many calls to the function *XAllocColor* returning all pixel values from `1` to `(map_entries - 1)`. For a visual class of *DirectColor,* the processing for *AllocAll* simulates a call to the function *XAllocColorPlanes,* returning a pixel value of zero and mask values the same as the *red_mask, green_mask,* and *blue_mask* members in *visual.*

The *visual* structure should be as returned from the *DefaultVisual* macro, *XMatchVisualInfo,* or *XGetVisualInfo.* The *red_mask, green_mask,* and *blue_mask* members specify which bits of the pixel value are allocated to each primary color. The *map_entries* member specifies the number of color map entries.

ERRORS

BadMatch	Didn't use *AllocNone* for *StaticColor, StaticGrey* or *TrueColor.* *visual* type not supported on screen.
BadAlloc	
BadValue	
BadWindow	

SEE ALSO

XCopyColormapAndFree, XFreeColormap, XGetStandardColormap, XInstallColormap, XUninstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XCreateFontCursor — create a cursor from standard cursor font.

SYNOPSIS

```
#include <X11/cursorfont.h>
Cursor XCreateFontCursor(display, shape)
  Display *display;
  unsigned int shape;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
shape	Specifies which character in the standard cursor font should be used for the cursor.

DESCRIPTION

X provides a set of standard cursor shapes in a special font named "cursorfont". Programs are encouraged to use this interface for their cursors, as the font can be customized for the individual display type and swapped between clients.

The hotspot comes from the information stored in the font. The initial colors of the cursor are black for the *foreground* and white for the *background*. *XRecolorCursor* can be used to change the colors of the cursor to those desired.

For further information about cursors and their shapes in fonts refer to the *GSE Programmer's Guide.*

ERRORS

BadAlloc
BadMatch
BadValue

SEE ALSO

XDefineCursor, XUndefineCursor, XCreateGlyphCursor, XCreatePixmapCursor, XFreeCursor, XRecolorCursor, XQueryBestCursor, XQueryBestSize.

NAME

XCreateGC — create new graphics context for a drawable.

SYNOPSIS

```
GC XCreateGC(display, d, valuemask, values)
  Display *display;
  Drawable d;
  unsigned long valuemask;
  XGCValues *values;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
valuemask	Specifies the components in the graphics context. This argument indicates which values are to be set using information in the *values* structure.
values	Specifies a pointer to an *XGCValues* structure which will provide components for the new GC.

DESCRIPTION

This function creates a new GC, replacing the old one if there was one. The specified components of the new graphics context in *valuemask* are set to the values passed in the *values* argument. Unset components default as follows:

Component	Value
function:	GXcopy
plane_mask:	all ones
foreground:	0
background:	1
line_width:	0
line_style:	LineSolid
cap_style:	CapButt
join_style:	JoinMiter
fill_style:	FillSolid
fill_rule:	EvenOddRule
arc_mode:	ArcPieSlice

tile:	`Pixmap of unspecified size filled with foreground pixel`
stipple:	`Pixmap of unspecified size filled with ones`
ts_x_origin:	`0`
ts_y_origin:	`0`
font:	`<implementation dependent>`
subwindow_mode:	`ClipByChildren`
graphics_exposures:	`True`
clip_x_origin:	`0`
clip_y_origin:	`0`
clip_mask:	`None`
dash_offset:	`0`
dash_list:	`4 (i.e., the list [4, 4])`

STRUCTURES

```
typedef struct {
  int function;                   /* logical operation */
  unsigned long plane_mask;       /* plane mask */
  unsigned long foreground;       /* foreground pixel */
  unsigned long background;       /* background pixel */
  int line_width;                 /* line width */
  int line_style;                 /* LineSolid, LineOnOffDash, LineDoubleDash */
  int cap_style;                  /* CapNotLast, CapButt, CapRound, CapProjectin
  int join_style;                 /* JoinMiter, JoinRound, JoinBevel */
  int fill_style;                 /* FillSolid, FillTiled, FillStippled */
  int fill_rule;                  /* EvenOddRule, WindingRule */
  int arc_mode;                   /* ArcPieSlice, ArcChord */
  Pixmap tile;                    /* tile pixmap for tiling operations */
  Pixmap stipple;                 /* stipple 1 plane pixmap for stipping */
  int ts_x_origin;                /* offset for tile or stipple operations */
  int ts_y_origin;
  Font font;                      /* default text font for text operations */
  int subwindow_mode;             /* ClipByChildren, IncludeInferiors */
  Bool graphics_exposures;        /* generate events on XCopyArea, XCopyPlane */
  int clip_x_origin;              /* origin for clipping */
  int clip_y_origin;
  Pixmap clip_mask;               /* bitmap clipping; other calls for rects */
  int dash_offset;                /* patterned/dashed line information */
  char dashes;
} XGCValues;
```

```
#define GCFunction              (1L<<0)
#define GCPlaneMask             (1L<<1)
#define GCForeground            (1L<<2)
#define GCBackground            (1L<<3)
#define GCLineWidth             (1L<<4)
#define GCLineStyle             (1L<<5)
#define GCCapStyle              (1L<<6)
#define GCJoinStyle             (1L<<7)
#define GCFillStyle             (1L<<8)
#define GCFillRule              (1L<<9)
#define GCTile                  (1L<<10)
#define GCStipple               (1L<<11)
#define GCTileStipXOrigin       (1L<<12)
#define GCTileStipYOrigin       (1L<<13)
#define GCFont                  (1L<<14)
#define GCSubwindowMode         (1L<<15)
#define GCGraphicsExposures     (1L<<16)
#define GCClipXOrigin           (1L<<17)
#define GCClipYOrigin           (1L<<18)
#define GCClipMask              (1L<<19)
#define GCDashOffset            (1L<<20)
#define GCDashList              (1L<<21)
#define GCArcMode               (1L<<22)
```

ERRORS

BadAlloc
BadDrawable
BadFont
BadMatch
BadPixmap
BadValue

SEE ALSO

XChangeGC, XCopyGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XCreateGlyphCursor — create a cursor from font glyphs.

SYNOPSIS

```
Cursor XCreateGlyphCursor(display, source_font, mask_font,
source_char, mask_char, foreground_color,   background_color)
  Display *display;
  Font source_font, mask_font;
  unsigned int source_char, mask_char;
  XColor *foreground_color;
  XColor *background_color;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
source_font	Specifies the font glyph for the cursor.
mask_font	Specifies the mask font. Optional.
source_char	Specifies the index into the cursor shape font.
mask_char	Specifies the index into the mask shape font. Optional.
foreground_color	Specifies the red, green, and blue (RGB) values for the foreground.
background_color	Specifies the red, green, and blue (RGB) values for the background.

DESCRIPTION

XCreateGlyphCursor is similar to *XCreatePixmapCursor*, but the source and mask bitmaps are obtained from separate font glyphs. The mask font and character are optional. If *mask_char* is not specified, all pixels of the source are displayed.

The origin of the character is the hotspot of the created cursor. In other words, the x offset for the hotspot is the left-bearing for the source character, and the y offset is the ascent.

The origins of the source and mask (if it is defined) glyphs are positioned coincidently and define the hotspot. The source and mask need not have the same bounding box metrics, and there is no restriction on the placement of the hotspot relative to the bounding boxes.

Note that *source_char* and *mask_char* are of type `unsigned int`, not of type `XChar2b`. For two-byte matrix fonts, the 16-bit value should be formed with the *byte1* member in the most significant byte and the *byte2* member in the least significant byte.

You can free the fonts with *XFreeFont* if they are no longer needed after creating the glyph cursor.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                       /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadAlloc

BadFont

BadValue *source_char* not defined in *source_font*.
mask_char not defined in *mask_font* (if *mask_font* defined).

SEE ALSO

XDefineCursor, XUndefineCursor, XCreateFontCursor, XCreatePixmapCursor, XFreeCursor, XRecolorCursor, XQueryBestCursor, XQueryBestSize.

NAME

XCreateImage — allocate memory for an XImage structure.

SYNOPSIS

```
#include <X11/Xutil.h>
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
visual	Specifies a pointer to the visual.
depth	Specifies the depth of the image.
format	Specifies the format for the image. Pass one of these constants: *XYPixmap*, or *ZPixmap*.
offset	Specifies the number of pixels beyond the first address of a scanline where the image actually begins. This is useful if the image is not aligned on an even addressable boundary.
data	Specifies a pointer to the image data.
width	Specifies the width (in pixels) of the image.
height	Specifies the height (in pixels) of 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*.

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 value of 0 here, Xlib assumes that the scanlines are contiguous in memory and thus calculates the value of *bytes_per_line* itself.

DESCRIPTION

XCreateImage allocates the memory needed for an *XImage* structure for the specified display and visual. This function does not allocate space for the image itself. Rather, it initializes the structure with "default" values and returns a pointer to the *XImage* structure. The red, green and blue mask values are defined for Z format images only and are derived from the *Visual* structure passed in. Refer to the *GSE Programmer's Guide* for a description of images.

SEE ALSO

XDestroyImage, XPutImage, XGetImage, XSubImage, XGetSubImage, XAddPixel, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XCreatePixmap — create a pixmap.

SYNOPSIS

```
Pixmap XCreatePixmap(display, d, width, height, depth)
  Display *display;
  Drawable d;
  unsigned int width, height;
  unsigned int depth;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable. May be an *InputOnly* window.
width *height*	Specify the width and height. These dimensions define the width and height of the pixmap. The values must be non-zero.
depth	Specifies the depth of the pixmap. The depth must be supported by the root of the specified drawable.

DESCRIPTION

XCreatePixmap creates a *pixmap* resource and returns its Pixmap ID. The initial contents of the pixmap are undefined.

The server uses the *drawable* argument to determine which screen the pixmap is stored on. The pixmap can only be used on this screen. The pixmap can only be used with other drawables of the same depth, except in *XCopyPlane*.

A bitmap is a single plane pixmap. There is no separate bitmap type in X version 11.

If this routine returns 0, there was insufficient space for the pixmap.

ERRORS

BadAlloc

BadDrawable

BadValue	*width* or *height* is zero.
	depth is not supported by root window.

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XCreatePixmapCursor — create a cursor from two bitmaps.

SYNOPSIS

```
Cursor XCreatePixmapCursor(display, source, mask,
foreground_color, background_color, x_hot, y_hot)
  Display *display;
  Pixmap source;
  Pixmap mask;
  XColor *foreground_color;
  XColor *background_color;
  unsigned int x_hot, y_hot;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
source	Specifies the shape of the source cursor. Pixmap of depth 1.
mask	Specifies the bits of the cursor that are to be displayed (the mask or stipple). Pixmap of depth 1.
foreground_color	Specifies the red, green, and blue (RGB) values for the foreground.
background_color	Specifies the red, green, and blue (RGB) values for the background.
x_hot *y_hot*	These coordinates indicate the hot spot relative to the source's origin, and must be a point within the source.

DESCRIPTION

XCreatePixmapCursor creates a cursor and returns a cursor ID. Foreground and background RGB values must be specified using *foreground_color* and *background_color*, even if the server only has a monochrome screen. The *foreground_color* is used for the one bits in the source, and the background is used for the zero bits. Both source and mask (if specified) must have depth one, but can have any root. The mask pixmap defines the shape of the cursor; that is, the one bits in the mask define which source pixels will be displayed. If no mask is given, all pixels of the source are displayed. The mask, if present, must be the same size as source.

The pixmaps can be freed immediately if no further explicit references to them are to be made.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                    /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadAlloc

BadMatch

BadPixmap

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XCreatePixmapFromBitmapData — create a pixmap with depth from bitmap data.

SYNOPSIS

Pixmap XCreatePixmapFromBitmapData(*display*, *drawable*, *data*, *width*, *height*, *fg*, *bg*, *depth*)
  Display **display*;
  Drawable *drawable*;
  char **data*;
  unsigned int *width*, *height*;
  unsigned long *fg*, *bg*;
  unsigned int *depth*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure, returned from *XOpenDisplay*.
drawable	Specifies a drawable ID which indicates which screen the pixmap is to be used on.
data	Specifies the data in bitmap format.
width *height*	Specify the width and height in pixels of the pixmap to create.
fg *bg*	Specifies the foreground and background pixel values to use.
depth	Specifies the depth of the pixmap. Must be valid on the screen specified by *drawable*.

DESCRIPTION

XCreatePixmapFromBitmapData creates a pixmap of the given depth using bitmap data and foreground and background pixel values.

The following format for the data is assigned by default, where the variables are members of the *XImage* structure described in the *GSE Programmer's Guide*.

```
format=XYPixmap
bit_order=LSBFirst
byte_order=LSBFirst
bitmap_unit=8
```

```
bitmap_pad=8
xoffset=0
no extra bytes per line
```

XCreatePixmapFromBitmapData creates an image from the data and uses *XPutImage* to place the data into the pixmap. For example:

```
#define gray_width 16
#define gray_height 16
#define gray_x_hot 8
#define gray_y_hot 8
static char gray_bits[] =
  {
  0xf81f, 0xe3c7, 0xcff3, 0x9ff9,
  0xbffd, 0x33cc, 0x7ffe, 0x7ffe,
  0x7e7e, 0x7ffe, 0x37ec, 0xbbdd,
  0x9c39, 0xcff3, 0xe3c7, 0xf81f/* example data */
  };
unsigned long foreground, background;
unsigned int depth;

/* open display, determine colors and depth */

Pixmap XCreatePixmapFromBitmapData(display, window, gray_bits, gray_width,
       gray_height, foreground, background, depth);
```

If you want to use data of a different format, it is straightforward to write a routine that does this yourself. See the Xlib code or the *MakePixmap* routine described in the *GSE Programmer's Guide* for an example.

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XCreateRegion — create a new empty region.

SYNOPSIS

```
Region XCreateRegion()
```

DESCRIPTION

XCreateRegion creates a new region of undefined size. *XPolygonRegion* can be used to create a region with defined shape and size. Many of the functions that perform operations on regions can also create regions.

For a description of Regions refer to the *GSE Programmer's Guide*.

STRUCTURES

```
typedef struct _XREGION *Region;/* opaque reference
to region type */
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XCreateSimpleWindow — creates an unmapped *InputOutput* window.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
parent	Specifies the parent window ID. Must be an *InputOutput* window.
x *y*	Specify the x and y coordinates of the top left outside corner of the new window's border relative to the inside of the parent window's border.
width *height*	Specify the width and height of the new window. These are the inside dimensions, not including the new window's borders, which are entirely outside of the window. Must be nonzero. Any part of the window that extends outside its parent window is clipped.
border_width	Specifies the width, in pixels, of the new window's border.
border	Specifies the pixel value for the border of the window.
background	Specifies the pixel value for the background of the window.

DESCRIPTION

XCreateSimpleWindow creates an unmapped *InputOutput* subwindow of the specified parent window. Use *XCreateWindow* to set the attributes to create an *InputOnly* window while creating a window.

XCreateSimpleWindow returns the window ID of the created window. The new window is placed on top of the stacking order relative to its siblings. Note that the window is unmapped when it is created—use *XMapWindow* to display it. This function generates a *CreateNotify* event.

The initial conditions of the window are as follows:

The window inherits its depth, class, and visual from its parent. All other window attributes have their default values.

All properties have undefined values.

The new window will not have a cursor defined; the cursor will be that of the window's parent until the cursor attribute is set with *XDefineCursor*.

If no background or border is specified, *CopyFromParent* is implied.

ERRORS

BadAlloc	
BadMatch	
BadValue	*width* and/or *height* is zero.
BadWindow	Specified parent is an *InputOnly* window.

SEE ALSO

XCreateWindow, XDestroySubwindows, XDestroyWindow.

NAME

XCreateWindow — create a window and set attributes.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
parent	Specifies the parent window. Parent must be *InputOutput* if class of window created is to be *InputOutput*.
x *y*	Specify the x and y coordinates. These coordinates are the top left outside corner of the new window's borders relative to the inside of the parent window's borders. (*x* = 0, *y* = 0) is the origin of the parent window.
width *height*	Specify the width and height. These are the new window's inside dimensions. These dimensions do not include the new window's borders, which are entirely outside of the window. Must be nonzero, otherwise *XCreateWindow* generates a *BadValue* error.
border_width	Specifies the width, in pixels, of the new window's border. Must be zero for *InputOnly* windows, otherwise a *BadMatch* error is returned.

depth	Specifies the depth of the window, not necessarily the same as the parent's depth. A depth of zero for class *InputOutput* or *CopyFromParent* means the depth is taken from the parent.
class	Specifies the new window's class. Pass one of these constants: *InputOutput, InputOnly,* or *CopyFromParent.*
visual	Specifies the visual type. *CopyFromParent* is valid.
valuemask	Specifies which window attributes are defined in the *attributes* argument. If *valuemask* is 0, the rest is ignored, and *attributes* is not referenced. This mask is the inclusive OR of the valid attribute mask bits.
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.

DESCRIPTION

To create an unmapped subwindow for a specified parent window from an application, you can use *XCreateWindow* or *XCreateSimpleWindow*. *XCreateWindow* is a more general function that allows you to set specific window attributes when you create it. If you do not want to set specific attributes when you create a window, use *XCreateSimpleWindow*, which creates a window that inherits its attributes from its parent. *XCreateSimpleWindow* creates *InputOutput* windows only.

XCreateWindow returns the window ID of the created window. *XCreateWindow* causes the X server to generate a *CreateNotify* event. The newly created window is placed on top of its siblings in the stacking order.

Extension packages may define other classes of windows. *XCreateWindow* returns the window ID of the created window and generates a *CreateNotify* event.

The visual should be *DefaultVisual* or one returned by *XGetVisualInfo* or *XMatchVisualInfo*.

STRUCTURES

```
/*
 * Data structure for setting window attributes.
 */
typedef struct {
  Pixmap background_pixmap;    /* background or None or ParentRelative */
  unsigned long background_pixel;/* background pixel */
  Pixmap border_pixmap;        /* border of the window */
  unsigned long border_pixel;  /* border pixel value */
  int bit_gravity;             /* one of bit gravity values */
  int win_gravity;             /* one of the window gravity values */
  int backing_store;           /* NotUseful, WhenMapped, Always */
  unsigned long backing_planes;/* planes to be preseved if possible */
  unsigned long backing_pixel; /* value to use in restoring planes */
  Bool save_under;             /* should bits under be saved (popups) */
  long event_mask;             /* set of events that should be saved */
  long do_not_propagate_mask;  /* set of events that should not propagate */
  Bool override_redirect;      /* boolean value for override-redirect */
  Colormap colormap;           /* colormap to be associated with window */
  Cursor cursor;               /* cursor to be displayed (or None) */
} XSetWindowAttributes;

/* Window attributes for CreateWindow and ChangeWindowAttributes */

/* Definitions for valuemask argument */

#define CWBackPixmap           (1L<<0)
#define CWBackPixel            (1L<<1)
#define CWBorderPixmap         (1L<<2)
#define CWBorderPixel          (1L<<3)
#define CWBitGravity           (1L<<4)
#define CWWinGravity           (1L<<5)
#define CWBackingStore         (1L<<6)
#define CWBackingPlanes        (1L<<7)
#define CWBackingPixel         (1L<<8)
#define CWOverrideRedirect     (1L<<9)
#define CWSaveUnder            (1L<<10)
#define CWEventMask            (1L<<11)
#define CWDontPropagate        (1L<<12)
#define CWColormap             (1L<<13)
#define CWCursor               (1L<<14)
```

ERRORS

BadAlloc	Attribute besides *win_gravity, event_mask, cursor, do_not_propagate_mask,* or *override_redirect* specified for *InputOnly.*
BadColor	*depth* non-zero for *InputOnly.*
BadCursor	Parent of *InputOutput* is *InputOnly.*
BadMatch	*border_width* is non-zero for *InputOnly.*
BadPixmap	*depth* not supported on screen for *InputOutput.*
BadValue	*width* and/or *height* is zero.
BadWindow	*visual* type not supported on screen (either *InputOnly* or *InputOutput*).

SEE ALSO

XCreateSimpleWindow, XDestroySubwindows, XDestroyWindow.

NAME

XDefineCursor — assign a cursor to a window.

SYNOPSIS

```
XDefineCursor(display, w, cursor)
  Display *display;
  Window w;
  Cursor cursor;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
cursor	Specifies the cursor. The function displays this cursor when the pointer is in the specified window. Pass **None** to have the parent's cursor displayed in the window. If it is the root window, then the default cursor is displayed.

DESCRIPTION

Sets the cursor attribute of a window, so that the specified cursor is shown whenever this window is visible and the pointer is inside. If *XDefineCursor* is not called, the parent's cursor is used by default.

ERRORS

BadAlloc
BadCursor
BadWindow

SEE ALSO

XUndefineCursor, XCreateFontCursor, XCreateGlyphCursor, XCreatePixmapCursor, XFreeCursor, XRecolorCursor, XQueryBestCursor, XQueryBestSize.

NAME

XDeleteAssoc — delete an entry from an association table.

SYNOPSIS

```
XDeleteAssoc(display, table, x_id)
  Display *display;
  XAssocTable *table;
  XID x_id;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
table	Specifies the assoc table.
x_id	Specifies the X resource ID.

DESCRIPTION

This function is provided for compatibility with X Version 10. To use it you must include the file <*X11/X10.h*> and link with the library **-loldX**.

XDeleteAssoc deletes an association in an *XAssocTable* keyed on its *XID*. Redundant deletes (and deletes of non-existent *XID*'s) are meaningless and cause no problems. Deleting associations in no way impairs the performance of an *XAssocTable*.

STRUCTURES

```
typedef struct {
  XAssoc *buckets;            /* pointer to first bucket in array */
  int size;                   /* table size (number of buckets) */
}XAssocTable;
```

SEE ALSO

XCreateAssocTable, XDestroyAssocTable, XLookUpAssoc, XMakeAssoc.

NAME

XDeleteContext — delete context entry for given window and type.

SYNOPSIS

```
int XDeleteContext(display, w, context)
  Display *display;
  Window w;
  XContext context;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window with which the data is associated.
context	Specifies the context type to which the data belongs.

DESCRIPTION

XDeleteContext deletes the entry for the given window and type from the context data structure defined in <*X11/Xutil.h*>. This returns *XCNOENT* if the context could not be found, or zero if it succeeds.

Refer to the *GSE Programmer's Guide* for a description of context management.

STRUCTURES

```
typedef int XContext;
```

SEE ALSO

XFindContext, XSaveContext, XUniqueContext.

NAME

XDeleteModifiermapEntry — delete an entry from an *XModifierKeymap* structure.

SYNOPSIS

XModifierKeymap ***XDeleteModifiermapEntry** (*modmap*, *keysym_entry*, *modifier*)
  XModifierKeymap **modmap*;
  KeyCode *keysym_entry*;
  int *modifier*;

ARGUMENTS

modmap	Specifies a pointer to an *XModifierKeymap* structure.
keysym_entry	Specifies the *KeyCode* of the key to be deleted from *modmap*.
modifier	Specifies the modifier you no longer want mapped to the keycode specified in *keysym_entry*. This should be one of the constants: *ShiftMapIndex, LockMapIndex, ControlMapIndex, Mod1MapIndex, Mod2MapIndex, Mod3MapIndex, Mod4MapIndex,* or *Mod5MapIndex.*

DESCRIPTION

XDeleteModifiermapEntry returns an *XModifierKeymap* structure suitable for calling *XSetModifierMapping,* in which the specified keycode is deleted from the set of keycodes that is mapped to the specified modifier (like Shift or Control). *XDeleteModifiermapEntry* does not change the mapping itself.

This function is normally used by calling *XGetModifierMapping* to get a pointer to the current *XModifierKeymap* structure for use as the *modmap* argument to *XDeleteModifiermapEntry.*

Note that the structure pointed to by *modmap* is freed by *XDeleteModifiermapEntry.* It should not be freed or otherwise used by applications.

For a description of the modifier map, see *XSetModifierMapping.*

STRUCTURES

```
typedef struct {
  int max_keypermod;          /* server's max number of keys per modifier */
  KeyCode *modifiermap;       /* an 8 by max_keypermod array of
                               * keycodes to be used as modifiers */
} XModifierKeymap;

#define ShiftMapIndex         0
#define LockMapIndex          1
#define ControlMapIndex       2
#define Mod1MapIndex          3
#define Mod2MapIndex          4
#define Mod3MapIndex          5
#define Mod4MapIndex          6
#define Mod5MapIndex          7
```

SEE ALSO

InsertModifiermapEntry, XGetModifierMapping, XSetModifierMapping, XNewModifiermap, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XRefreshKeyboardMapping, XLookupString.

NAME

XDeleteProperty — delete a window property.

SYNOPSIS

```
XDeleteProperty(display, w, property)
  Display *display;
  Window w;
  Atom property;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose property you want to delete.
property	Specifies the property atom.

DESCRIPTION

XDeleteProperty deletes a window property, so that it no longer contains any data. Its atom, specified by *property*, still exists after the call so that it can be used again later by any application that knows the ID of the window the property is defined on. If the property was defined on the specified window, *XDeleteProperty* generates a *PropertyNotify* event.

Refer to the *GSE Programmer's Guide.*

ERRORS

BadAtom
BadWindow

SEE ALSO

XSetStandardProperties, XGetFontProperty, XRotateWindowProperties, XChangeProperty, XGetWindowProperty, XListProperties, XGetAtomName, XInternAtom.

NAME

XDestroyAssocTable — free the memory allocated for association table.

SYNOPSIS

```
XDestroyAssocTable(table)
  XAssocTable *table;
```

ARGUMENTS

table	Specifies the assoc table.

DESCRIPTION

This function is provided for compatibility with X Version 10. To use it you must include the file *<X11/X10.h>* and link with the library `-loldX`.

Using an *XAssocTable* after it has been destroyed will have unpredictable and probably disastrous consequences.

STRUCTURES

```
typedef struct {
  XAssoc *buckets;                 / *pointer to first bucket in array */
  int size;                        / *table size (number of buckets) */
}XAssocTable;
```

SEE ALSO

XCreateAssocTable, XDeleteAssoc, XLookUpAssoc, XMakeAssoc.

NAME

XDestroyImage — deallocate memory associated with an image.

SYNOPSIS

```
int XDestroyImage(ximage)
    XImage  *ximage;
```

ARGUMENTS

ximage	Specifies a pointer to the image.

DESCRIPTION

XDestroyImage deallocates the memory associated with an *XImage* structure. This memory includes both the memory holding the *XImage* structure, and the memory holding the actual image data.

SEE ALSO

XPutImage, XGetImage, XCreateImage, XSubImage, XGetSubImage, XAddPixel, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XDestroyRegion — deallocate storage associated with a region.

SYNOPSIS

```
XDestroyRegion(r)
  Region r;
```

ARGUMENTS

r	Specifies the region.

DESCRIPTION

XDestroyRegion frees the memory associated with a region.

Refer to the *GSE Programmer's Guide* for a description of regions.

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XEqualRegion, XClipBox.

NAME

XDestroySubwindows — destroy all subwindows of a window.

SYNOPSIS

```
XDestroySubwindows(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

This function destroys all descendants of the specified window, in bottom to top stacking order.

XDestroySubwindows generates exposure events on *w*, if any mapped subwindows were actually destroyed. This is much more efficient than deleting many subwindows one at a time, as much of the work need only be performed once for all of the windows rather than for each window. It also saves multiple exposure events on the windows about to be destroyed. The subwindows should never again be referenced.

XCloseDisplay automatically destroys all windows that have been created by that client on the specified display (unless called after a fork system call--see note under *XCloseDisplay*).

ERRORS

BadWindow

SEE ALSO

XCreateSimpleWindow, XCreateWindow, XDestroyWindow.

NAME

XDestroyWindow — unmap and destroy a window and all subwindows.

SYNOPSIS

```
XDestroyWindow(display, window)
  Display *display;
  Window window;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
window	Specifies the window ID.

DESCRIPTION

If *window* is mapped, an *UnmapWindow* request is performed automatically. The window and all inferiors are then destroyed, and a *DestroyNotify* event is generated for each window. The ordering of the *DestroyNotify* events is such that for any given window, *DestroyNotify* is generated on all inferiors of the window before being generated on the window itself. The ordering among siblings and across subhierarchies is not otherwise constrained.

The windows should never again be referenced. Destroying a mapped window will generate exposure events on other windows that were obscured by the windows being destroyed.

No windows are destroyed if you try to destroy the root window.

XDestroyWindow may generate *EnterWindow* events if *window* was mapped and contained the pointer.

ERRORS

BadWindow

SEE ALSO

XCreateSimpleWindow, XCreateWindow, XDestroySubwindows.

NAME

XDisableAccessControl — prevent modification to the host access list.

SYNOPSIS

```
XDisableAccessControl(display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XDisableAccessControl prevents any other client that subsequently connects to the server from changing the access control list.

ERRORS

BadAccess

SEE ALSO

XAddHost, XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XEnableAccessControl, XSetAccessControl.

NAME

XDisplayName — reports the display name when connecting to that display fails.

SYNOPSIS

```
char *XDisplayName(string)
  char *string;
```

ARGUMENTS

string	Specifies the character string.

DESCRIPTION

XDisplayName is normally used to report the name of the display the program attempted to open with *OpenDisplay*. This is necessary because X error handling begins only after the connection to the server succeeds. If a *NULL* string is specified, *XDisplayName* looks in the environment for the display and returns the display name that the user was requesting. Otherwise, *XDisplayName* returns its own argument. This makes it easier to report to the user precisely which display the program attempted to open.

SEE ALSO

XGetErrorDatabaseText, XGetErrorText, XSetErrorHandler, XSetIOErrorHandler, XSynchronize, XSetAfterFunction.

NAME

XDraw — draw polyline or curve between vertex list (from X10).

SYNOPSIS

```
Status XDraw(display, d, gc, vlist, vcount)
  Display *display;
  Drawable d;
  GC gc;
  Vertex *vlist;
  int vcount;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
gc	Specifies the graphics context.
vlist	Specifies a pointer to the list of vertices which indicate what to draw.
vcount	Specifies how many vertices are in *vlist*.

DESCRIPTION

This function is provided for compatibility with X Version 10. To use it you must include the file *<X11/X10.h>* and link with the library **-loldX**.

XDraw achieves the effects of the V10 *XDraw, XDrawDashed,* and *XDrawPatterned* functions.

XDraw draws an arbitrary polygon or curve. The figure drawn is defined by the specified list of vertices (*vlist*). The points are connected by lines as specified in the flags each the *Vertex* structure.

The *Vertex* structure contains an *x,y* coordinate and a bitmask called *flags* that specifies the drawing parameters.

The x and y elements of *Vertex* are the coordinates of the vertex that are relative to either the previous vertex (if *VertexRelative* is 1) or the upper left inside corner of the drawable (if *VertexRelative* is 0). If *VertexRelative* is 0 the coordinates are said to be absolute. The first vertex must be an absolute vertex.

If the *VertexDontDraw* bit is 1, no line or curve is drawn from the previous vertex to this one. This is analogous to picking up the pen and moving to another place before drawing another line.

If the *VertexCurved* bit is 1, a spline algorithm is used to draw a smooth curve from the previous vertex, through this one, to the next vertex. Otherwise, a straight line is drawn from the previous vertex to this one. It makes sense to set *VertexCurved* to 1 only if a previous and next vertex are both defined (either explicitly in the array, or through the definition of a closed curve--see below.)

It is permissible for *VertexDontDraw* bits and *VertexCurved* bits to both be *1*. This is useful if you want to define the previous point for the smooth curve, but you do not want an actual curve drawing to start until this point.

If *VertexStartClosed* bit is *1*, then this point marks the beginning of a closed curve. This vertex must be followed later in the array by another vertex whose absolute coordinates are identical and which has *VertexEndClosed* bit of *1*. The points in between form a cycle for the purpose of determining predecessor and successor vertices for the spline algorithm.

XDraw uses the following graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_s_origin, ts_y_origin, dash_offset,* and *dash_list.*

Status of 0 on failure.

STRUCTURES

```
typedef struct _Vertex {
  short x,y;
  unsigned short flags;
} Vertex;

/* defined constants for use as flags */

#define VertexRelative        0x0001     /* else absolute */
#define VertexDontDraw        0x0002     /* else draw */
#define VertexCurved          0x0004     /* else straight */
#define VertexStartClosed     0x0008     /* else not */
#define VertexEndClosed       0x0010     /* else not */
```

SEE ALSO

XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawArc — draws an arc fitting inside a rectangle.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates relative to the drawable. These coordinates specify the upper left corner of the rectangle that contains the arc.
width *height*	Specify the width and height. These are the major and minor axes of the arc.
angle1	Specifies the start of the arc relative to the three o'clock position from the center. Angles are specified in degrees, multiplied by 64 (360 * 64 is a complete circle).
angle2	Specifies the path and extent of the arc relative to the start of the arc. Angles are specified in degrees, multiplied by 64 (360 * 64 is a complete circle).

DESCRIPTION

XDrawArc draws a circular or elliptical arc. An arc is specified by a rectangle and two angles. The *x* and *y* coordinates are relative to the origin of the drawable and define the upper left corner of the rectangle. The center of the circle or ellipse is the center of the rectangle; the major and minor axes are specified by the *width* and *height*, respectively. The angles are signed integers in degrees multiplied by 64, with positive indicating counterclockwise motion and negative indicating clockwise motion, truncated to a maximum of 360 degrees. The start of the arc is specified by *angle1* relative to the three o'clock position from the center; the path and extent of the arc is specified by *angle2* relative to the start of the arc.

By specifying one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio.

XDrawArc uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dash_list. XDrawArc* is not affected by the tile or stipple in the GC.

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawArcs — draw multiple arcs.

SYNOPSIS

```
XDrawArcs(display, d, gc, arcs, narcs)
  Display  *display;
  Drawable d;
  GC gc;
  XArc  *arcs;
  int narcs;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
gc	Specifies the graphics context.
arcs	Specifies a pointer to an array of arcs.
narcs	Specifies the number of arcs in the array.

DESCRIPTION

This is the plural version of *XDrawArc*. See *XDrawArc* for details of drawing a single arc.

The arcs are drawn in the order listed in the *arcs* array. For any given arc, no pixel is drawn more than once. If arcs intersect, pixels will be drawn multiple times. If the last point in one arc coincides with the first point in the following arc, the two arcs will join correctly according to the GC. If the first point in the first arc coincides with the last point in the last arc, the two arcs will join correctly according to the GC.

By specifying one axis to be zero, a horizontal or vertical line can be drawn. Angles are computed based solely on the coordinate system and ignore the aspect ratio.

For any given arc, no pixel is drawn more than once. If two arcs join correctly and if *line_width* is greater than zero and the arcs intersect, no pixel is drawn more than once. Otherwise, the intersecting pixels of intersecting arcs are drawn multiple times. Specifying an arc with one endpoint and a clockwise extent draws the same pixels as specifying the other endpoint and an equivalent counterclockwise extent, except as it affects joins.

If the last point in one arc coincides with the first point in the following arc, the two arcs will join correctly. If the first point in the first arc coincides with the last point in the last arc, the two arcs will join correctly.

XDrawArcs uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dash_list. XDrawArcs* is not affected by the tile or stipple in the GC.

The following is a technical explanation of the points drawn by *XDrawArcs.* For an arc specified as `[x,y,width,height,angle1,angle2]`, the origin of the major and minor axes is at `[x+(width/2),y+(height/2)]`, and the infinitely thin path describing the entire circle or ellipse intersects the horizontal axis at `[x,y+(height/2)]` and `[x+width,y+(height/2)]` and intersects the vertical axis at `[x+(width/2),y]` and `[x+(width/2),y+height]`. These coordinates can be fractional. That is, they are not truncated to discrete coordinates. The path should be defined by the ideal mathematical path. For a wide line with line width *line_width,* the bounding outlines for filling are given by the infinitely thin paths describing the arcs:

```
[x+dx/2, y+dy/2, width-dx, height-dy, angle1, angle2]
```

and

```
[x-line_width/2, y-line_width/2, width+line_width, height+line_width,
        angle1, angle2]
```

where

```
dx=min(line_width,width)
dy=min(line_width,height)
```

If `(height != width)` the angles must be specified in the effectively skewed coordinate system of the ellipse (for a circle, the angles and coordinate systems are identical). The relationship between these angles and angles expressed in the normal coordinate system of the screen (as measured with a protractor) is as follows.

```
skewed-angle = atan(tan(normal-angle) * width/height) + adjust
```

The skewed-angle and normal-angle are expressed in radians (rather than in degrees scaled by 64) in the range `[0,2*PI)`, and where atan returns a value in the range `[-PI/2,PI/2]`, and where adjust is:

```
0       for normal-angle in the range [0,PI/2)
PI      for normal-angle in the range [PI/2,(3*PI)/2)
2*PI    for normal-angle in the range [(3*PI)/2,2*PI)
```

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
  short angle1, angle2;        /*  Degrees * 64  */
} XArc;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawFilled — draw filled polygon or curve from vertex list (from V10).

SYNOPSIS

```
Status XDrawFilled(display, d, gc, vlist, vcount)
  Display *display;
  Drawable d;
  GC gc;
  Vertex *vlist;
  int vcount;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
gc	Specifies the graphics context.
vlist	Specifies a pointer to the list of vertices.
vcount	Specifies how many vertices are in *vlist*.

DESCRIPTION

This function is provided for compatibility with X Version 10. To use it you must include the file *<X11/X10.h>* and link with the library **-loldX**. *XDrawFilled* achieves the effects of the V10 *XDrawTiled* and *XDrawFilled* functions.

XDrawFilled draws arbitrary polygons or curves, according to the same rules as *XDraw*, and then fills them.

XDrawFilled uses the following graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_s_origin, ts_y_origin, dash_offset, dash_list, fill_style* and *fill_rule*.

XDrawFilled returns status of 0 on failure.

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawImageString — draw 8-bit image text characters.

SYNOPSIS

```
XDrawImageString(display, d, gc, x, y, string, length)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  char *string;
  int length;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These coordinates define the baseline starting position for the image text character and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the *string* argument.

DESCRIPTION

XDrawImageString draws a string, but unlike *XDrawString* it can draw both the foreground and the background of the characters, if the GC is set accordingly.

XDrawImageString uses these graphics context components: *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* The *function* and *fill_style* defined in *gc* are ignored; the effective function is *GXcopy* and the effective *fill_style* is *FillSolid.*

XDrawImageString first fills a destination rectangle with the *background* pixel defined in *gc,* and then paints the text with the *foreground* pixel. The upper left corner of the filled rectangle is at *[x, y - font_ascent],* the width is *overall->width,* and the height is *Xascent + descent.*

The *overall->width, ascent,* and *descent* are as would be returned by *XQueryTextExtents* using *gc* and *string.*

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XDrawImageString16 — draw 16-bit image text characters.

SYNOPSIS

```
XDrawImageString16(display, d, gc, x, y, string, length)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  XChar2b *string;
  int length;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the image text character and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the *string* argument.

DESCRIPTION

XDrawImageString16 draws a string, but unlike *XDrawString16* it can draw both the foreground and the background of the characters, if the GC is set accordingly.

XDrawImageString16 uses these graphics context components: *plane_mask, foreground, background, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. The *function* and *fill_style* defined in *gc* are ignored; the effective function is *GXcopy* and the effective *fill_style* is *FillSolid*.

XDrawImageString16 first fills a destination rectangle with the *background* pixel defined in *gc,* and then paints the text with the *foreground* pixel. The upper left corner of the filled rectangle is at *[x, y - font_ascent]*, the width is *overall->width,* and the height is *ascent + descent*.

The *overall->width, ascent,* and *descent* are as would be returned by *XQueryTextExtents16* using *gc* and *string*.

STRUCTURES

```
typedef struct {
  unsigned char byte1;
  unsigned char byte2;
} XChar2b;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XDrawLine — draw a line between two points.

SYNOPSIS

```
XDrawLine(display, d, gc, x1, y1, x2, y2)
  Display *display;
  Drawable d;
  GC gc;
  int x1, y1, x2, y2;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x1 *y1* *x2* *y2*	Specify the end points of the line relative to the drawable origin. *XLine* connects point (*x1*, *y1*) to point (*x2*, *y2*).

DESCRIPTION

XDrawLine uses the components of the specified graphics context to draw a line between two points in the specified drawable. No pixel is drawn more than once.

XDrawLine uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. *XDrawLine* also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dash_list*.

XDrawLine is not affected by *tile* or *stipple* in the GC.

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawLines — draw multiple connected lines.

SYNOPSIS

```
XDrawLines(display, d, gc, points, npoints, mode)
  Display *display;
  Drawable d;
  GC gc;
  XPoint *points;
  int npoints;
  int mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
points	Specifies a pointer to an array of points.
npoints	Specifies the number of points in the array.
mode	Specifies the coordinate mode. Pass either *CoordModeOrigin* or *CoordModePrevious*.

DESCRIPTION

XDrawLines does the following:

- Draws lines connecting each point in the list (*points* array) to the next point in the list. The lines are drawn in the order listed in the *points* array. For any given line, no pixel is drawn more than once. If thin (zero line width) lines intersect, pixels will be drawn multiple times. If the first and last points coincide, the first and last lines will join correctly. If wide lines intersect, the intersecting pixels are drawn only once, as though the entire multi-line request were a single filled shape.
- Uses the components of the specified graphics context to draw multiple connected lines in the specified drawable. Specifically, *XDrawLines* uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and

dash_list.

The *mode* argument may have two values:

- *CoordModeOrigin* indicates that all points are relative to the drawable's origin.
- *CoordModePrevious* indicates that all points after the first are relative to the previous point. (The first point is always relative to the drawable's origin.)

XDrawLines is not affected by the tile or stipple in the GC.

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
} XPoint;
```

ERRORS

BadDrawable
BadGC
BadMatch
BadValue

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawPoint — draw a point.

SYNOPSIS

```
XDrawPoint(display, d, gc, x, y)
  Display  *display;
  Drawable d;
  GC gc;
  int x, y;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates of the point, relative to the corner of the drawable.

DESCRIPTION

XDrawPoint uses the *foreground* pixel and *function* components of the graphics context to draw a single point into the specified drawable. *XDrawPoint* uses these graphics context components: *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* Use *XDrawPoints* to draw multiple points.

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawPoints — draw multiple points.

SYNOPSIS

```
XDrawPoints(display, d, gc, points, npoints, mode)
  Display *display;
  Drawable d;
  GC gc;
  XPoint *points;
  int npoints;
  int mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
points	Specifies a pointer to an array of *XPoint* structures containing the positions of the points.
npoints	Specifies the number of points to be drawn.
mode	Specifies the coordinate mode. *CoordModeOrigin* treats all coordinates as relative to the origin, while *CoordModePrevious* treats all coordinates after the first as relative to the previous point, while the first is still relative to the origin.

DESCRIPTION

XDrawPoints uses the *foreground* pixel and *function* components of the graphics context to draw one or more points into the specified drawable.

XDrawPoints uses these graphics context components: *function, plane_mask, foreground, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.*

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
} XPoint;
```

ERRORS

BadDrawable
BadGC
BadMatch
BadValue

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawRectangle — draw outline of rectangle.

SYNOPSIS

```
XDrawRectangle(display, d, gc, x, y, width, height)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  unsigned int width, height;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These coordinates define the upper left corner of the rectangle relative to the drawable's origin.
width *height*	Specify the width and height. These dimensions define the outline of the rectangle.

DESCRIPTION

XDrawRectangle draws the outline of the rectangle by using the *x* and *y* coordinates, *width* and *height*, and graphics context you specify. Specifically, *XDrawRectangle* uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_x_origin, ts_y_origin, dash_offset,* and *dash_list.*

XDrawRectangle is not affected by the tile or stipple in the GC. For the specified rectangle, no pixel is drawn more than once.

STRUCTURE

```
typedef struct {
  short x, y;
  unsigned short width, height;
} XRectangle;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawRectangles — draw the outlines of multiple rectangles.

SYNOPSIS

```
XDrawRectangles(display, d, gc, rectangles, nrectangles)
  Display *display;
  Drawable d;
  GC gc;
  XRectangle rectangles[] ;
  int nrectangles;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
rectangles	Specifies a pointer to an array of rectangles.
nrectangles	Specifies the number of rectangles in the array.

DESCRIPTION

XDrawRectangles draws the outlines of the specified rectangles by using the position and size values in the array of rectangles. The x and y coordinates of each rectangle are relative to the drawable's origin, and define the upper left corner of the rectangle. This function uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, join_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. *XDrawRectangles* also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dash_list*.

The rectangles are drawn in the order listed. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, pixels are drawn multiple times.

XDrawRectangles is not affected by *tile* or *stipple* in the GC.

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
  unsigned short width, height;
} XRectangle;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawSegments — draw multiple disjoint lines.

SYNOPSIS

```
XDrawSegments (display, d, gc, segments, nsegments)
  Display *display;
  Drawable d;
  GC gc;
  XSegment *segments;
  int nsegments;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
segments	Specifies a pointer to an array of segments.
nsegments	Specifies the number of segments in the array.

DESCRIPTION

XDrawSegments draws multiple line segments into the specified drawable. Each line is specified by a pair of points, so the line may be connected or disjoint.

For each segment, *XDrawSegments* draws a line between (*x1, y1*) and (*x2, y2*). The lines are drawn in the order listed in *segments*. For any given line, no pixel is drawn more than once. If lines intersect, pixels will be drawn multiple times. The lines will be drawn separately, without regard to the *join_style*.

XDrawSegments uses these graphics context components: *function, plane_mask, line_width, line_style, cap_style, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. *XDrawSegments* also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin, ts_y_origin, dash_offset,* and *dash_list*.

XDrawSegments is not affected by the tile or stipple in the GC.

STRUCTURES

```
typedef struct {
  short x1, y1, x2, y2;
} XSegment;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XDrawString — draw 8-bit text string, foreground only.

SYNOPSIS

```
XDrawString(display, d, gc, x, y, string, length)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  char *string;
  int length;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These coordinates define the baseline starting position for the character and are relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in the string argument.

DESCRIPTION

XDrawString draws the given string into a drawable using the *foreground* only to draw set bits in the font. It does not affect any other pixels in the bounding box for each character.

The *y* coordinate defines the baseline row of pixels while the *x* coordinate is the point for measuring the *lbearing, rbearing,* and *width* from.

XDrawString uses these graphics context components: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, tile, stipple, ts_x_origin,* and *ts_y_origin.* Each character image, as defined by the font in *gc,* is treated as an additional mask for a fill operation on the drawable.

ERRORS

BadDrawable
BadFont
BadGC
BadMatch

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XDrawString16 — draw two-byte text strings.

SYNOPSIS

```
XDrawString16(display, d, gc, x, y, string, length)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  XChar2b *string;
  int length;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the character, and are relative to the origin of the specified drawable.
string	Specifies the character string. Characters are two bytes wide.
length	Specifies the number of characters in the string argument.

DESCRIPTION

XDrawString16 draws a string in the foreground pixel value without drawing the surrounding pixels.

The *y* coordinate defines the baseline row of pixels while the *x* coordinate is the point for measuring the *lbearing, rbearing,* and *width* from.

XDrawString16 uses these graphics context components: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. This function also uses these graphics context mode-dependent components: *foreground, tile, stipple, ts_x_origin,* and *ts_y_origin*. Each character image, as defined by the font in *gc,* is treated as an additional mask for a fill operation on the drawable.

STRUCTURES

```
typedef struct {
  unsigned char byte1;
  unsigned char byte2;
} XChar2b;
```

ERRORS

BadDrawable
BadFont
BadGC
BadMatch

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XDrawText — draw 8-bit polytext characters.

SYNOPSIS

```
XDrawText(display, d, gc, x, y, items, nitems)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  XTextItem *items;
  int nitems;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the initial string, relative to the origin of the specified drawable.
items	Specifies a pointer to an array of text items.
nitems	Specifies the number of text items in the array.

DESCRIPTION

XDrawText is capable of drawing multiple strings and changing fonts between strings. Each *XTextItem* structure contains a string, the number of characters in the string, the *delta* offset from the starting position for the string, and the font. Each text item is processed in turn. The font in each *XTextItem* is stored in the specified GC and used for subsequent text. If the *XTextItem.font* is *None,* the font in the GC is used for drawing and is not changed. Switching between fonts with different drawing directions is permitted.

The *delta* in each *XTextItem* specifies the change in horizontal position before the string is drawn. The delta is always added to the character origin and is not dependent on the draw direction of the font. For example, if *x = 40, y = 20,* and *items[0].delta = 8,* the string specified by *items[0].chars* would be drawn starting at *x = 48, y = 20.* The *delta* for the second string begins at the *rbearing* of the last character in the first string. A negative *delta* would tend to overlay subsequent strings on the end of the previous string.

Only the pixels selected in the font are drawn (the *background* member of the GC is not used).

XDrawText uses the following elements in the specified GC: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, tile, stipple, ts_x_origin,* and *ts_y_origin.*

STRUCTURES

```
typedef struct {
  char *chars;                /* pointer to string */
  int nchars;                 /* number of characters */
  int delta;                  /* delta between strings */
  Font font;                  /* font to print it in, None
                               * doesn't change */
} XTextItem;
```

ERRORS

BadDrawable
BadFont
BadGC
BadMatch

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XDrawText16 — draw 16-bit polytext strings.

SYNOPSIS

```
XDrawText16(display, d, gc, x, y, items, nitems)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  XTextItem16 *items;
  int nitems;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x	
y	Specify the x and y coordinates. These coordinates define the baseline starting position for the initial string relative to the origin of the specified drawable.
items	Specifies a pointer to an array of text items using 2-byte characters.
nitems	Specifies the number of text items in the array.

DESCRIPTION

XDrawText16 is capable of drawing multiple strings and changing fonts between strings. Each *XTextItem* structure contains a string, the number of characters in the string, the *delta* offset from the starting position for the string, and the font. Each text item is processed in turn. The font in each *XTextItem* is stored in the specified GC and used for subsequent text. If the *XTextItem16.font* is *None,* the font in the GC is used for drawing and is not changed. Switching between fonts with different drawing directions is permitted.

The *delta* in each *XTextItem* specifies the change in horizontal position before the string is drawn. The delta is always added to the character origin and is not dependent on the draw direction of the font. For example, if *x = 40, y = 20,* and *items[0].delta = 8,* the string specified by *items[0].chars* would be drawn starting at *x = 48, y = 20.* The *delta* for the second string begins at the *rbearing* of the last character in the first string.

A negative *delta* would tend to overlay subsequent strings on the end of the previous string.

Only the pixels selected in the font are drawn (the *background* member of the GC is not used).

XDrawText16 uses the following elements in the specified GC: *function, plane_mask, fill_style, font, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, tile, stipple, ts_x_origin,* and *ts_y_origin.*

Note that the *chars* member of the *XTextItem16* structure is of type *XChar2b,* rather than of type *char* as it is in the *XTextItem* structure. For fonts defined with linear indexing rather than two-byte matrix indexing, the X server will interpret each member of the *XChar2b* structure as a 16-bit number that has been transmitted most significant byte first. In other words, the *byte1* member of the *XChar2b* structure is taken as the most significant byte.

STRUCTURES

```
typedef struct {
  XChar2b *chars;                 /* 2 byte characters */
  int nchars;                     /* number of characters */
  int delta;                      /* delta between strings */
  Font font;                      /* font to print it in, None
                                   * doesn't change */
} XTextItem16;

typedef struct {                  /* normal 16 bit characters are two bytes */
    unsigned char byte1;
    unsigned char byte2;
} XChar2b;
```

ERRORS

BadDrawable
BadFont
BadGC
BadMatch

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XEmptyRegion — determine if region is empty.

SYNOPSIS

```
int XEmptyRegion(r)
  Region r;
```

ARGUMENTS

r	Specifies the region.

DESCRIPTION

XEmptyRegion will return non-zero if the region is empty.

STRUCTURES

```
/*
 * opaque reference to Region data type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XEnableAccessControl — enable changes to the access control list.

SYNOPSIS

```
XEnableAccessControl (display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XEnableAccessControl allows other clients that connect to the server after this call to modify the access control list. If access has not been disabled with *XDisableAccessControl* or *XSetAccessControl*, this does nothing.

As always, the access control list can only be modified by clients connected to a display on the host whose list is to be modified. In other words, you must have access to change access.

SEE ALSO

XAddHost, XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XDisableAccessControl, XSetAccessControl.

NAME

XEqualRegion — determine if two regions have the same size, offset, and shape.

SYNOPSIS

```
int XEqualRegion(r1, r2)
  Region r1, r2;
```

ARGUMENTS

r1	
r2	Specify the two regions you want to compare.

DESCRIPTION

XEqualRegion returns non-zero if the two regions are identical, i.e. have the same offset, size and shape.

Regions are located using an offset from an arbitrarily chosen point (the "region origin") which is common to all regions. It is up to the application to interpret the location of the region relative to a drawable.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XClipBox.

NAME

XEventsQueued — check the number of events in the event queue.

SYNOPSIS

```
int XEventsQueued(display, mode)
    Display *display;
    int mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure, returned from *XOpenDisplay*.
mode	Specifies whether the output buffer is flushed if there are no events in Xlib's queue. You can specify one of these constants: *QueuedAlready, QueuedAfterFlush, QueuedAfter-Reading.*

DESCRIPTION

XEventsQueued checks whether events are queued. If there are events in Xlib's queue, the routine returns immediately to the calling routine. Its return value is the number of events regardless of *mode.*

mode specifies what happens if no events are found on Xlib's queue.

- If *mode* is *QueuedAlready,* and there are no events in the queue, *XEventsQueued* returns 0 (it does not flush the output buffer or attempt to read more events from the connection).
- If *mode* is *QueuedAfterFlush,* and there are no events in the queue, *XEventsQueued* flushes the output buffer, attempts to read more events out of the application's connection, and returns the number read.
- If *mode* is *QueuedAfterReading,* and there are no events in the queue, *XEventsQueued* attempts to read more events out of the application's connection without flushing the output buffer and returns the number read.

Note that *XEventsQueued* always returns immediately without I/O if there are events already in the queue.

XEventsQueued with mode *QueuedAfterFlush* is identical in behavior to *XPending. XEventsQueued* with mode *QueuedAlready* is identical to the *QLength* macro.

For more information, refer to the *GSE Programmer's Guide.*

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XSynchronize, XSendEvent, QLength, XPending.

NAME

XFetchBuffer — return data from cut buffer.

SYNOPSIS

```
char *XFetchBuffer(display, nbytes, buffer)
  Display *display;
  int *nbytes;                /* RETURN */
  int buffer;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
nbytes	Returns the number of bytes in *buffer* returned by **XFetchBuffer**. If there is no data in the buffer, *nbytes* is set to 0.
buffer	Specifies which buffer you want data from.

DESCRIPTION

XFetchBuffer returns data from one of the 8 buffers provided for inter-client communication. If the buffer contains data, *XFetchBuffer* returns the number of bytes in *nbytes,* otherwise it returns *NULL* and sets *nbytes* to 0. The appropriate amount of storage is allocated and the pointer returned; the client must free this storage when finished with it. Note that the cut buffer does not necessarily contain text, so it may contain embedded null bytes and may not terminate with a null byte.

Selections are the preferred communication scheme.

SEE ALSO

XStoreBuffer, XStoreBytes, XFetchBytes, XRotateBuffers.

NAME

XFetchBytes — return data from cut buffer 0.

SYNOPSIS

```
char *XFetchBytes(display, nbytes)
  Display *display;
  int *nbytes;                    /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
nbytes	Returns the number of bytes in the string returned by *XFetchBytes*. If there is no data in the buffer, *nbytes* is set to 0.

DESCRIPTION

XFetchBytes returns data from cut buffer 0 of the 8 buffers provided for inter-client communication. If the buffer contains data, *XFetchBytes* returns the number of bytes in *nbytes*, otherwise it returns *NULL* and sets *nbytes* to 0. The appropriate amount of storage is allocated and the pointer returned; the client must free this storage when finished with it. Note that the cut buffer does not necessarily contain text, so it may contain embedded null bytes and may not terminate with a null byte.

Use *XFetchBuffer* to fetch data from any specified cut buffer.

Selections are the preferred communication method.

SEE ALSO

XStoreBuffer, XStoreBytes, XFetchBuffer, XRotateBuffers.

NAME

XFetchName — get window name (WM_NAME property).

SYNOPSIS

```
Status XFetchName(display, w, window_name)
  Display *display;
  Window w;
  char **window_name;        /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose name you want a pointer set to.
window_name	Returns a pointer to the window name, which will be a null-terminated string. If the WM_NAME property has not been set for this window, *XFetchName* sets *windowname* to *NULL*. When finished with it, a client must free the name string using *XFree*.

DESCRIPTION

XFetchName returns the current value of the WM_NAME property for the specified window. *XFetchName* return value is non-zero if it succeeds, and 0 if the property has not been set for the argument window.

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XFillArc — fill an arc.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These coordinates are relative to the origin of the drawable and specify the upper left corner of the rectangle.
width *height*	Specify the width and height. These are the major and minor axes of the arc.
angle1	Specifies the start of the arc relative to the three o'clock position from the center. Angles are specified in degrees, multiplied by 64.
angle2	Specifies the path and extent of the arc relative to the start of the arc. Angles are specified in degrees, multiplied by 64.

DESCRIPTION

XFillArc fills an arc according to the *arc_mode* in the GC. The *x, y, width,* and *height* arguments specify the bounding box for the arc. See *XDrawArc* for the description of how this bounding box is used to compute the arc. Some, but not all, of the pixels drawn with *XDrawArc* will be drawn with *XFillArc* with the same arguments.

The arc forms one boundary of the area to be filled. The other boundary is determined by the *arc_mode* in the GC. If the *arc_mode* in the GC is

ArcChord, the single line segment joining the endpoints of the arc is used. If *ArcPieSlice,* the two line segments joining the endpoints of the arc with the center point are used.

XFillArc uses these graphics context components: *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin.*

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XFillArcs — fill multiple arcs.

SYNOPSIS

```
XFillArcs(display, d, gc, arcs, narcs)
  Display *display;
  Drawable d;
  GC gc;
  XArc  *arcs;
  int narcs;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
arcs	Specifies a pointer to an array of arc definitions.
narcs	Specifies the number of arcs in the array.

DESCRIPTION

For each arc, *XFillArcs* fills the region closed by the specified arc and one or two line segments, depending on the *arc_mode* specified in the GC. It does not draw the complete outlines of the arcs, but some pixels may overlap.

The arc forms one boundary of the area to be filled. The other boundary is determined by the *arc_mode* in the GC. If the *arc_mode* in the GC is *ArcChord,* the single line segment joining the endpoints of the arc is used. If *ArcPieSlice,* the two line segments joining the endpoints of the arc with the center point are used. The arcs are filled in the order listed in the array. For any given arc, no pixel is drawn more than once. If regions intersect, pixels will be drawn multiple times.

XFillArcs use these graphics context components: *function, plane_mask, fill_style, arc_mode, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin.*

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
  unsigned short width, height;
  short angle1, angle2;        /*  Degrees * 64  */
} XArc;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XFillPolygon — fill a polygon.

SYNOPSIS

```
XFillPolygon(display, d, gc, points, npoints, shape, mode)
  Display *display;
  Drawable d;
  GC gc;
  XPoint *points;
  int npoints;
  int shape;
  int mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
points	Specifies a pointer to an array of points.
npoints	Specifies the number of points in the array.
shape	Specifies an argument that helps the server to improve performance. Pass the last constant in this list that is valid for the polygon to be filled: *Complex, Nonconvex,* or *Convex.*
mode	Specifies the coordinate mode. Pass either *CoordModeOrigin* or *CoordModePrevious.*

DESCRIPTION

XFillPolygon fills the region closed by the specified path. Some but not all of the path itself will be drawn. The path is closed automatically if the last point in the list does not coincide with the first point. No pixel of the region is drawn more than once.

The *mode* argument affects the interpretation of the points that define the polygon:

- *CoordModeOrigin* indicates that all points are relative to the drawable's origin.
- *CoordModePrevious* indicates that all points after the first are relative to the previous point. (The first point is always relative to the drawable's origin.)

The *shape* argument allows the fill routine to optimize its performance given tips on the configuration of the area.

- *Complex* indicates the path may self-intersect. The *fill_rule* of the GC must be consulted to determine which areas are filled. Refer to the *GSE Programmer's Guide* for a discussion of the fill rules *EvenOddRule* and *WindingRule.*
- *Nonconvex* indicates the path does not self-intersect, but the shape is not wholly convex. If known by the client, specifying *Nonconvex* instead of *Complex* may improve performance. If you specify *Nonconvex* for a self-intersecting path, the graphics results are undefined.
- *Convex* indicates the path is wholly convex. This can improve performance even more, but if the path is not convex, the graphics results are undefined.

XFillPolygon uses these graphics context components when filling the polygon area: *function, plane_mask, fill_style, fill_rule, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these mode-dependent components of the GC: *foreground, background, tile, stipple, ts_x_origin,* and *ts_y_origin.*

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
} XPoint;
```

ERRORS

BadDrawable
BadGC
BadMatch
BadValue

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XFillRectangle — fill rectangular area.

SYNOPSIS

```
XFillRectangle(display, d, gc, x, y, width, height)
  Display *display;
  Drawable d;
  GC gc;
  int x, y;
  unsigned int width, height;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates. These coordinates are relative to the origin of the drawable and specify the upper left corner of the rectangle.
width *height*	Specify the dimensions of the rectangle to be filled.

DESCRIPTION

XFillRectangle fills the rectangular area in the specified drawable using the *x* and *y* coordinates, *width* and *height* dimensions, and graphics context you specify. *XFillRectangle* draws some but not all of the path drawn by *XDrawRectangle* with the same arguments.

XFillRectangle uses these graphics context components: *function, plane_mask, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask*. This function also uses these graphics context components depending on the *fill_style*: *foreground, background tile, stipple, ts_x_origin,* and *ts_y_origin*.

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangles, XClearArea, XClearWindow.

NAME

XFillRectangles — fill multiple rectangular areas.

SYNOPSIS

```
XFillRectangles(display, d, gc, rectangles, nrectangles)
  Display *display;
  Drawable d;
  GC gc;
  XRectangle *rectangles;
  int nrectangles;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
gc	Specifies the graphics context.
rectangles	Specifies a pointer to an array of rectangles.
nrectangles	Specifies the number of rectangles in the array.

DESCRIPTION

XFillRectangles fills multiple rectangular areas in the specified drawable using the graphics context.

The *x* and *y* coordinates of each rectangle are relative to the drawable's origin, and define the upper left corner of the rectangle. The rectangles are drawn in the order listed. For any given rectangle, no pixel is drawn more than once. If rectangles intersect, the intersecting pixels will be drawn multiple times.

XFillRectangles uses these graphics context components: *function, plane_mask, fill_style, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context components depending on the *fill_style*: *foreground, background tile, stipple, ts_x_origin,* and *ts_y_origin.*

STRUCTURES

```
typedef struct {
  short x, y;
  unsigned short width, height;
  unsigned short width, height;
} XRectangle;
```

ERRORS

BadDrawable
BadGC
BadMatch

SEE ALSO

XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDrawRectangle, XDrawRectangles, XDrawSegments, XCopyArea, XCopyPlane, XFillArc, XFillArcs, XFillPolygon, XFillRectangle, XFillRectangles, XClearArea, XClearWindow.

NAME

XFindContext — get data from context manager (not graphics context).

SYNOPSIS

```
int XFindContext(display, w, context, data)
  Display *display;
  Window w;
  XContext context;
  caddr_t *data;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window with which the data is associated.
context	Specifies the context type to which the data corresponds.
data	Returns the data.

DESCRIPTION

XFindContext gets data that has been assigned to the specified window and context ID. The context manager is used to associate data with windows for use within an application.

This application should have called *XUniqueContext* to get a unique ID, and then *XSaveContext* to save the data into the array. The meaning of the data is indicated by the context ID, but is completely up to the client.

XFindContext returns *XCNOENT* (a nonzero error code) if the context could not be found and zero (0) otherwise.

STRUCTURES

```
typedef int XContext
```

SEE ALSO

XDeleteContext, XSaveContext, XUniqueContext.

NAME

XFlush — flush the output buffer (display all queued requests).

SYNOPSIS

```
XFlush(display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XFlush sends to the display ("flushes") all output requests that have been buffered but not yet sent.

Flushing is done automatically when input is read if no matching events are in Xlib's queue (with *XPending, XNextEvent,* or *XWindowEvent*), or when a call is made that gets information from the server (such as *XQueryPointer, XGetFontInfo*) so *XFlush* is seldom needed. It is used when the buffer must be flushed before any of these calls are reached.

SEE ALSO

XSync

NAME

XForceScreenSaver — turn screen saver on or off.

SYNOPSIS

```
XForceScreenSaver(display, mode)
  Display *display;
  int mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
mode	Specifies whether screen saver is active or reset. The possible modes are: *ScreenSaverActive* or *ScreenSaverReset*.

DESCRIPTION

XForceScreenSaver resets or activates the screen saver. If the specified mode is *ScreenSaverActive* and the screen saver currently is disabled, the screen saver is activated, even if the screen saver had been disabled with a timeout of zero (0). This means that the screen may go blank or have some random change take place to save the phosphors. If the specified mode is *ScreenSaverReset* and the screen saver currently is enabled, the screen is returned to normal, the screen saver is deactivated and the activation timer is reset to its initial state (as if device input had been received). *Expose* events may be generated on all visible windows if the server cannot save the entire screen contents.

ERRORS

BadValue

SEE ALSO

XActivateScreenSaver, XResetScreenSaver, XGetScreenSaver, XSetScreenSaver.

NAME

XFree — free specified in-memory data created by an Xlib function.

SYNOPSIS

```
XFree(data)
  char *data;
```

ARGUMENTS

data	Specifies a pointer to the data that is to be freed.

DESCRIPTION

XFree is a general purpose routine for freeing data allocated by Xlib calls.

SEE ALSO

XOpenDisplay, XCloseDisplay, XNoOp, DefaultScreen.

NAME

XFreeColormap — delete colormap and install the default colormap.

SYNOPSIS

```
XFreeColormap(display, cmap)
  Display *display;
  Colormap cmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap to delete.

DESCRIPTION

XFreeColormap destroys the colormap, unless it is the default colormap for a screen. That is, it not only uninstalls *cmap* from the hardware colormap if it is installed, but also frees the associated memory and removes all trace.

XFreeColormap performs the following processing:

- If *cmap* is an installed map for a screen, it uninstalls the colormap and installs the default if not already installed.
- If *cmap* is defined as the colormap attribute for a window (by *XCreateWindow* or *XChangeWindowAttributes*), it changes the colormap associated with the window to the constant *None,* generates a *ColormapNotify* event, and frees the colormap. The colors displayed with a colormap of *None* are server-dependent as the default colormap is normally used.

ERRORS

BadColor

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XGetStandardColormap, XInstallColormap, XUninstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XFreeColors — free colormap cells or planes.

SYNOPSIS

```
XFreeColors(display, cmap, pixels, npixels, planes)
  Display *display;
  Colormap cmap;
  unsigned long pixels[];
  int npixels;
  unsigned long planes;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
pixels	Specifies an array of pixel values. These pixel values map to the cells in the specified colormap.
npixels	Specifies the number of pixels.
planes	Specifies the planes you want to free.

DESCRIPTION

XFreeColors frees the cells whose values are computed by ORing together subsets of the *planes* argument with each pixel value in the *pixels* array.

If the cells are read/write, they become available for reuse, unless they were allocated with *XAllocColorPlanes,* in which case all the related pixels may need to be freed before any become available.

If the cells were read-only, they become available only if this is the last client who had allocated those shared cells.

ERRORS

BadAccess	A colorcell allocated by client (either unallocated or allocated by another client).
BadColor	
BadValue	A pixel value is not a valid index into *cmap*.

If more than one pixel value is in error, the one reported is arbitrary.

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XFreeCursor — destroy a cursor.

SYNOPSIS

```
XFreeCursor(display, cursor)
  Display *display;
  Cursor cursor;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cursor	Specifies the cursor ID.

DESCRIPTION

XFreeCursor deletes the association between the cursor ID and the specified cursor. The cursor storage is freed when all other clients have freed it. Windows with their cursor attribute set to this cursor will be changed to *None* (which implies *CopyFromParent*). The specified cursor should not be referred to again or an error will be generated.

ERRORS

BadCursor

SEE ALSO

XDefineCursor, XUndefineCursor, XCreateFontCursor, XCreateGlyphCursor, XCreatePixmapCursor, XRecolorCursor, XQueryBestCursor, XQueryBestSize.

NAME

XFreeExtensionList — free memory allocated for list of installed extensions to X.

SYNOPSIS

```
XFreeExtensionList(list)
  char **list;
```

ARGUMENTS

list	Specifies the list of extensions returned from *XListExtensions.*

DESCRIPTION

XFreeExtensionList frees the memory allocated by *XListExtensions.*

SEE ALSO

XListExtensions, XQueryExtension.

NAME

XFreeFont — unload font and free storage for font structure.

SYNOPSIS

```
XFreeFont(display, font_struct)
  Display  *display;
  XFontStruct  *font_struct;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
font_struct	Specifies the storage associated with the font.

DESCRIPTION

XFreeFont frees the memory allocated for the *font_struct* font information structure (*XFontStruct*) filled by *XQueryFont* or *XLoadQueryFont*. *XFreeFont* frees all storage associated with the *font_struct* argument. Neither the data nor the font should be referenced again.

The font itself is unloaded if no other client has loaded it.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;       /*hook for extension to hang data*/
  Font fid;                 /*font id for this font*/
  unsigned direction;       /*hint about direction the font is painted*/
  unsigned min_char_or_byte2;/*first character*/
  unsigned max_char_or_byte2;/*last character*/
  unsigned min_byte1;       /*first row that exists*/
  unsigned max_byte1;       /*last row that exists*/
  Bool all_chars_exist;     /*flag if all characters have non-zero size*/
  unsigned default_char;    /*char to print for undefined character*/
  int n_properties;         /*how many properties there are*/
  XFontProp *properties;    /*pointer to array of additional properties*/
  XCharStruct min_bounds;   /*minimum bounds over all existing char*/
  XCharStruct max_bounds;   /*minimum bounds over all existing char*/
  XCharStruct *per_char;    /*first_char to last_char information*/
  int ascent;               /*logical extent above baseline for spacing*/
  int descent;              /*logical descent below baseline for spacing */
} XFontStruct;
```

ERRORS

BadFont

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreePath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XFreeFontInfo — free multiple font information arrays.

SYNOPSIS

```
XFreeFontInfo(names, info, actual_count)
  char **names;
  XFontStruct *info;
  int actual_count;
```

ARGUMENTS

names	Specifies a pointer to the list of font names that were returned by *XListFontsWithInfo*.
info	Specifies a pointer to the list of font information that was returned by *XListFontWithInfo*.
actual_count	Specifies the number of matched font names returned by *XListFontWithInfo*.

DESCRIPTION

XFreeFontInfo frees all resources allocated by *XListFontsWithInfo*. It does not unload the specified fonts.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;             /*hook for extension to hang data*/
  Font fid;                       /*font id for this font*/
  unsigned direction;             /*hint about direction the font is painted*/
  unsigned min_char_or_byte2;     /*first character*/
  unsigned max_char_or_byte2;     /*last character*/
  unsigned min_byte1;             /*first row that exists*/
  unsigned max_byte1;             /*last row that exists*/
  Bool all_chars_exist;           /*flag if all characters have non-zero size*/
  unsigned default_char;          /*char to print for undefined character*/
  int n_properties;               /*how many properties there are*/
  XFontProp *properties;          /*pointer to array of additional properties*/
  XCharStruct min_bounds;         /*minimum bounds over all existing char*/
  XCharStruct max_bounds;         /*minimum bounds over all existing char*/
  XCharStruct *per_char;          /*first_char to last_char information*/
  int ascent;                     /*logical extent above baseline for spacing*/
  int descent;                    /*logical descent below baseline for spacing*/
} XFontStruct;
```

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XListFonts, XListFontsWithInfo, XFreeFontNames, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XFreeFontNames — free font name array.

SYNOPSIS

```
XFreeFontNames (list)
  char  *list[];
```

ARGUMENTS

list	Specifies the array of font name strings to be freed.

DESCRIPTION

XFreeFontNames frees the array of strings returned by *XListFonts.*

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XFreeFontPath — free memory allocated by XGetFontPath.

SYNOPSIS

```
XFreeFontPath(list)
  char **list;
```

ARGUMENTS

list	Specifies the array of strings allocated by *XGetFontPath.*

DESCRIPTION

Frees the data used by the array of directories returned by *XGetFontPath.*

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XFreeGC — free a graphics context.

SYNOPSIS

```
XFreeGC(display, gc)
  Display *display;
  GC gc;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.

DESCRIPTION

Frees all memory associated with a graphics context, and removes the GC from the server and display hardware.

ERRORS

BadGC

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XFreeModifiermap — destroy and free keyboard modifier mapping table.

SYNOPSIS

```
XFreeModifiermap(modmap)
      XModifierKeymap  *modmap;
```

ARGUMENTS

modmap	Specifies a pointer to the *XModifierKeymap* structure.

DESCRIPTION

XFreeModifiermap frees the specified *XModifierKeymap* structure.

STRUCTURES

```
typedef struct {
        int max_keypermod;   /*server's max number of keys per modifie
        KeyCode *modifiermap;/*an 8 by max_keypermod array of
                                      * keycodes to be used as modifiers

} XModifierKeymap;
```

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XFreePixmap — free pixmap ID.

SYNOPSIS

```
XFreePixmap(display, pixmap)
  Display *display;
  Pixmap pixmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
pixmap	Specifies the pixmap.

DESCRIPTION

XFreePixmap disassociates a pixmap ID from its resource. If no other client has an ID for that resource, it is freed. The *Pixmap* should never be referenced again by this client. If it is, the ID will be unknown and a *BadPixmap* error will result.

ERRORS

BadPixmap

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XGContextFromGC — obtain the GContext (ID) associated with the specified GC.

SYNOPSIS

```
GContext XGContextFromGC(gc)
  GC gc;
```

ARGUMENTS

gc	Specifies the graphics context that you want the resource ID for.

DESCRIPTION

XGContextFromGC extracts the resource ID from the GC structure. Using the *gc* argument, *gc->gid*, does the same thing.

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XGeometry — calculate window geometry given user geometry string and default geometry.

SYNOPSIS

```
int XGeometry(display, screen, user_geom, default_geom, bwidth,
fwidth, fheight, xadder, yadder, x, y, width, height)
  Display *display;
  int screen;
  char *user_geom, *default_geom;
  unsigned int bwidth;
  unsigned int fwidth, fheight;
  int xadder, yadder;
  int *x, *y, *width, *height;/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
screen	Specifies which screen the window is on.
user_geom	Specifies the user- or program-supplied geometry string, perhaps incomplete.
default_geom	Specifies the default geometry string.
bwidth	Specifies the border width.
fheight *fwidth*	Specify the font height and width in pixels (increment size).
xadder *yadder*	Specify additional interior padding needed in the window.
width *height*	Return the window dimensions.
x *y*	Return the window placement.

DESCRIPTION

XGeometry returns the position and size of a window placement given a user-supplied geometry (allowed to be partial) and a default geometry. Each user-supplied specification is copied into the appropriate returned argument, unless it is not present, in which case the default specification is used. The default geometry should be complete while the user-supplied one may not be.

XGeometry is useful for processing command line and *~/.Xdefaults* options. These geometry strings are of the form:

=<*width*>x<*height*>{+-}<*xoffset*>{+-}<*yoffset*>

The "=" at the beginning of the string is now optional.

The *XGeometry* return value is a bitmask which indicates which values were present in *user_geom*. This bitmask is composed of the exclusive OR of the symbols *XValue, YValue, WidthValue, HeightValue, XNegative,* or *YNegative.*

If the function returns either *XValue* or *YValue,* you should place the window at the requested position. The border width (*bwidth*), size of the width and height increments (typically *fwidth* and *fheight*), and any additional interior space (*xadder* and *yadder*) are passed in to make it easy to compute the resulting size.

SEE ALSO

XParseGeometry, XTranslateCoordinates.

NAME

XGetAtomName — get name for atom.

SYNOPSIS

```
char *XGetAtomName(display, atom)
  Display *display;
  Atom atom;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
atom	Specifies the atom whose string name you want returned.

DESCRIPTION

An atom is a symbol (actually a number) identifying a property. *XGetAtomName* returns a string version of the atom name. *XA_WM_CLASS* (a symbol) is returned as "*XA_WM_CLASS*" (a string). If the specified atom name is not defined, *XGetAtomName* returns NULL.

XInternAtom performs the inverse function.

ERRORS

BadAtom

SEE ALSO

XSetStandardProperties, XGetFontProperty, XRotateWindowProperties, XDeleteProperty, XChangeProperty, XGetWindowProperty, XListProperties, XInternAtom.

NAME

XGetClassHint — get the WM_CLASS property of a window.

SYNOPSIS

```
Status XGetClassHint(display, w, class_hints)
  Display *display;
  Window w;
  XClassHint *class_hints;    /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
class_hints	Returns the *XClassHints* structure.

DESCRIPTION

XGetClassHint obtains the XA_WM_CLASS property for the specified window.

XGetClassHint returns a status of 0 on failure, non-zero on success.

The *XClassHint* structure returned contains `res_class`, which is the name of the client such as "emacs", and *res_name,* which is the first of the following that applies:

- command line option (*–rn name*)
- a specific environment variable (e.g., RESOURCE_NAME)
- the trailing component of `argv[0]`

STRUCTURES

```
typedef struct {
  char *res_name;
  char *res_class;
} XClassHint;
```

ERRORS

BadWindow

SEE ALSO

XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGetDefault — scan user preference file for program name and options.

SYNOPSIS

```
char *XGetDefault(display, program, option)
  Display *display;
  char *program;
  char *option;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
program	Specifies the program name to be looked for in *~/.Xdefaults.* The program name is usually *argv[0],* the first argument on the SYSTEM V/68 command line.
option	Specifies the option name or keyword. Lines containing both the *program* name and the *option* name will be matched.

DESCRIPTION

XGetDefault returns a character string containing the value at the end of line that contains both the *program* name and the *option* name specified. The strings returned by *XGetDefault* are owned by Xlib and should not be modified or freed by the client.

Lines in *~/.Xdefaults* look like this:

```
xterm.foreground:          #c0c0ff
xterm.geometry:            =81x28
xterm.saveLines:           256
xterm.font:                8x13
xterm.keyMapFile:          /usr/black/.keymap
xterm.activeIcon:          on
```

Upper and lower case is important. In some programs the standard is to capitalize only the second and successive words in each option, if any. In others, the first word is also capitalized.

Defaults are usually loaded into the RESOURCE_MANAGER property on the root window at login. If no such property exists, a resource file in the user's home directory is loaded. On a SYSTEM V/68-based system, this file is *$HOME/.Xdefaults*. After loading these defaults, `XGetDefault` merges additional defaults specified by the XENVIRONMENT

environment variable. If XENVIRONMENT is defined, it contains a full path name for the additional resource file. If XENVIRONMENT is not defined, **XGetDefault** looks for *$HOME/.Xdefaults*-name, where *name* specifies the name of the machine on which the application is running.

The first invocation of *XGetDefault* reads the defaults into memory so that subsequent requests are fast. Therefore, changes to the defaults files from the program will not be felt until the next invocation.

XGetDefault returns the value *NULL* if the option name specified in this argument does not exist for the program.

SEE ALSO

XAutoRepeatOff, XAutoRepeatOn, XBell, XGetKeyboardControl, XChangeKeyboardControl, XGetPointerControl.

NAME

XGetErrorDatabaseText — obtain error messages from the error data base.

SYNOPSIS

```
XGetErrorDatabaseText(display, name, message, default_string,
buffer, length)
  Display display;
  char *name, *message;
  char *default_string;
  char *buffer;                /* RETURN */
  int length;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
name	Specifies the name of the application.
message	Specifies the type of the error message. One of *XProtoError, XlibMessage,* or *XRequestMajor* (see Description below).
default_string	Specifies the default error message.
buffer	Returns the error description.
length	Specifies the size of the return buffer.

DESCRIPTION

XGetErrorDatabaseText returns a message from the error message database. Given *name* and *message* as keys, *XGetErrorDatabaseText* uses the resource manager to look up a string and returns it in the buffer argument. Xlib uses this function internally to look up its error messages. On a SYSTEM V/68-based system, the error message database is */usr/lib/XerrorDB.*

The *name* argument should generally be the name of your application. The *message* argument should indicate which type of error message you want. Three predefined *message* types are used by Xlib to report errors:

- *XProtoError*

The protocol error number is used as a string for the message argument.

- *XlibMessage*

These are the message strings that are used internally by the library.

- *XRequestMajor*

The major request protocol number is used for the message argument.

If no string is found in the error data base, *XGetErrorDatabaseText* returns the *default_string* that you specify to the buffer.

The string in *buffer* will be of length *length*.

SEE ALSO

XDisplayName, XGetErrorText, XSetErrorHandler, XSetIOErrorHandler, XSynchronize, XSetAfterFunction.

NAME

XGetErrorText — obtain description of error code.

SYNOPSIS

```
XGetErrorText(display, code, buffer, length)
  Display *display;
  int code;
  char *buffer;                 /* RETURN */
  int length;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
code	Specifies the error code for which you want to obtain a description.
buffer	Returns a pointer to the error description text.
length	Specifies the size of the buffer.

DESCRIPTION

XGetErrorText obtains textual descriptions of errors. *XGetErrorText* returns a pointer to a null-terminated string describing the specified error code with length *length*. This string is copied from static data and therefore may be freed. This routine allows extensions to the Xlib library to define their own error codes and error strings, which can be accessed easily.

SEE ALSO

XDisplayName, XGetErrorDatabaseText, XSetErrorHandler, XSetIOErrorHandler, XSynchronize, XSetAfterFunction.

NAME

XGetFontPath — get the current font search path.

SYNOPSIS

```
char **XGetFontPath(display, npaths)
  Display *display;
  int *npaths;             /* RETURN number of ele-
ments */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
npaths	Returns the number of strings in the font path array.

DESCRIPTION

XGetFontPath allocates and returns an array of strings containing the search path. The data in the font path should be freed when no longer needed.

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XGetFontProperty — get a font property given its atom.

SYNOPSIS

```
Bool XGetFontProperty(font_struct, atom, value)
  XFontStruct *font_struct;
  Atom atom;
  unsigned long *value;    /* RETURN */
```

ARGUMENTS

font_struct	Specifies the storage associated with the font.
atom	Specifies the atom associated with the property name you want returned.
value	Returns the value of the font property.

DESCRIPTION

XGetFontProperty returns the value of the specified font property, given the atom for that property. The function returns 0 if the atom was not defined, or 1 if was defined.

There are a set of predefined atoms for font properties which can be found in *<X11/Xatom.h>*. These atoms are listed and described in the *GSE Programmer's Guide*. This set contains the standard properties associated with a font. The predefined font properties are likely but not guaranteed to be present on any given server.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;          /*hook for extension to hang data*/
  Font fid;                    /*Font id for this font*/
  unsigned direction;          /*hint about direction the font is painted*/
  unsigned min_char_or_byte2;  /*first character*/
  unsigned max_char_or_byte2;  /*last character*/
  unsigned min_byte1;          /*first row that exists*/
  unsigned max_byte1;          /*last row that exists*/
  Bool all_chars_exist;        /*flag if all characters have non-zero size*/
  unsigned default_char;       /*char to print for undefined character*/
  int n_properties;            /*how many properties there are*/
  XFontProp *properties;       /*pointer to array of additional properties*/
  XCharStruct min_bounds;      /*minimum bounds over all existing char*/
  XCharStruct max_bounds;      /*minimum bounds over all existing char*/
  XCharStruct *per_char;       /*first_char to last_char information*/
```

```
    int ascent;                 /*logical extent above baseline for spacing*/
    int descent;                /*logical descent below baseline for spacing*/
  } XFontStruct;
```

SEE ALSO

XSetStandardProperties, XRotateWindowProperties, XDeleteProperty, XChangeProperty, XGetWindowProperty, XListProperties, XGetAtomName, XInternAtom.

NAME

XGetGeometry — obtain current geometry of drawable.

SYNOPSIS

```
Status XGetGeometry(display, d,  root, x, y, width, height,
border_width, depth)
  Display  *display;
  Drawable d;
  Drawable  *root;           /* RETURN */
  int *x,  *y;               /* RETURN */
  unsigned int *width, *height;/* RETURN */
  unsigned int *border_width;/* RETURN */
  unsigned int *depth;       /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, either a window or a pixmap.
root	Returns the root window ID of the specified window.
x *y*	Return the location of the drawable for a window. They are the upper left outer corner relative to its parent's origin. For pixmaps, these coordinates are always 0.
width *height*	Return the dimensions of the drawable. For a window, these return the inside size (not including the border). For a pixmap, they just return the size.
border_width	Returns the borderwidth, in pixels, of the window's border, if the drawable is a window. Returns 0 if the drawable is a pixmap.
depth	Returns the depth of the pixmap (bits per pixel for the object) The depth must be supported by the root of the specified drawable.

DESCRIPTION

This function gets complete information about the current geometry of a drawable.

XGetGeometry returns a status of 0 on failure or 1 when it succeeds.

ERRORS

BadDrawable

SEE ALSO

XGetWindowAttributes, XChangeWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap.

NAME

XGetIconName — get name to be displayed in icon.

SYNOPSIS

```
Status XGetIconName(display, w, icon_name)
  Display *display;
  Window w;
  char **icon_name;           /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose icon name you want to learn.
icon_name	Returns a pointer to the name to be displayed in the window's icon. The name should be a null-terminated string. If you never assigned a name to the window, *XGetIconName* sets this argument to *NULL*. When finished with it, a client must free the icon name string using *XFree*.

DESCRIPTION

XGetIconName reads the icon name property of a window. This function is primarily used by window managers to get the name to be written in that window's icon when they need to display that icon.

The *XGetIconName* return value is non-zero status if it succeeds, and 0 if no icon name has been set for the argument window.

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGetIconSizes — get preferred icon sizes.

SYNOPSIS

```
Status XGetIconSizes(display, w, size_list, count)
  Display *display;
  Window w;
  XIconSize **size_list;        /* RETURN */
  int *count;                   /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID (usually of the root window).
size_list	Returns a pointer to the size list.
count	Returns the number of items in the size list.

DESCRIPTION

XGetIconSizes returns zero if a window manager has not set icon sizes, and a non-zero status otherwise. This function should be called by all programs to find out what icon sizes are preferred by the window manager. The application should then use *XSetWMHints* to supply the window manager with an icon pixmap or window in one of the supported sizes.

STRUCTURES

```
typedef struct {
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
} XIconSize;

/* width_inc and height_inc provide the preferred
 * increment of sizes in the range from min_width
 * to max_width and min_height to max_height. */
```

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XStoreName, XSetIconSizes, XSetCommand.

NAME

XGetImage — place contents of rectangle from drawable into image.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable to get the data from.
x *y*	Specify the x and y coordinates. These coordinates define the upper left corner of the rectangle and are relative to the origin of the drawable.
width *height*	Specify the width and height. These arguments define the dimensions of the image.
plane_mask	Specifies a plane mask which indicates which planes are represented in image.
format	Specifies the format for the image. Pass either *XYPixmap* or *ZPixmap*.

DESCRIPTION

XGetImage provides a mechanism to perform a rudimentary screen dump.

XGetImage returns an *XImage* structure. This structure provides you with the contents of the specified rectangle of the drawable in the format you specify. If you specify the *XYPixmap* format, the function gets only the bit planes you passed to the *plane_mask* argument. If you specify the *ZPixmap* format, the function sets as zero the bits in all planes not specified in the *plane_mask* argument. The function performs no range checking on the values in *plane_mask*, and ignores extraneous bits.

XGetImage returns the depth of the image to the depth member of the *XImage* structure. The depth of the image is as specified when the drawable was created.

If the drawable is a pixmap, the specified rectangle must be completely inside the pixmap, or a *BadMatch* error will occur.

If the drawable is a window, the window must be mapped. It must also be the case that, if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen; otherwise, a *BadMatch* error will occur. The returned image will include any visible portions of inferiors or overlapping windows contained in the rectangle. The specified area can include the borders. The returned contents of visible regions of inferiors of different depth than the specified window are undefined.

If the window has a backing-store, the backing-store contents are returned for regions of the window that are obscured by noninferior windows. Otherwise, the return contents of such obscured regions are undefined. Also undefined are the returned contents of visible regions of inferiors of different depth than the specified window.

For *XYFormat format* data, the *bit_order* member of *XImage* specifies which bit order your server likes the data in.

If *XGetImage* fails for any reason, it returns NULL.

ERRORS

BadDrawable	
BadMatch	See Description above.
BadValue	

SEE ALSO

XDestroyImage, XPutImage, XCreateImage, XSubImage, XGetSubImage, XAddPixel, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XGetInputFocus — discover current input focus window.

SYNOPSIS

```
XGetInputFocus(display, focus, revert_to)
  Display *display;
  Window *focus;              /* RETURN */
  int *revert_to;             /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
focus	Returns the ID of the focus window, or one of the constants *PointerRoot* or *None*.
revert_to	Returns the window to which the focus would revert if the focus window became invisible. This is one of these constants: *RevertToParent, RevertToPointerRoot,* or *RevertTo-None*. Must not be a window ID.

DESCRIPTION

XGetInputFocus returns the current focus window and the window to which the focus would revert if the focus window became invisible.

It does not report the last focus change time. This is available only from events.

SEE ALSO

XSelectInput, XSetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XGetKeyboardControl — obtain list of current keyboard control values.

SYNOPSIS

```
XGetKeyboardControl(display, values)
  Display *display;
  XKeyboardState *values;    /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
values	Returns filled *XKeyboardState* structure.

DESCRIPTION

XGetKeyboardControl returns the current control values for the keyboard. For the LEDs, the least significant bit of led-mask corresponds to LED one, and each bit that is set to one in *led_mask* indicates an LED that is lit. **auto_repeats** is a bit vector; each bit that is set to one indicates that auto-repeat is enabled for the corresponding key. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7, with the least significant bit in the byte representing key 8N. *global_auto_repeat* is either *AutoRepeatModeOn* or *AutoRepeatModeOff*.

For the ranges of each member of *XKeyboardState*, see the routine that sets that value.

STRUCTURES

```
typedef struct {
  int key_click_percent;
  int bell_percent;
  unsigned int bell_pitch, bell_duration;
  unsigned long led_mask;
  int global_auto_repeat;
  char auto_repeats[32];
} XKeyboardState;
```

SEE ALSO

XGetDefault, XAutoRepeatOff, XAutoRepeatOn, XBell, XChangeKeyboardControl, XGetPointerControl.

NAME

XGetKeyboardMapping — return symbols for keycodes.

SYNOPSIS

KeySym *XGetKeyboardMapping(*display*, *first_keycode*, *keycode_count*, *keysyms_per_keycode*)
  Display **display*;
  KeyCode *first_keycode*;
  int *keycode_count*;
  int **keysyms_per_keycode*;   /* **RETURN** */

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
first_keycode	Specifies the first keycode that is to be returned.
keycode_count	Specifies the number of keycodes that are to be returned.
keysyms_per_keycode	Returns the number of keysyms per keycode.

DESCRIPTION

Starting with *first_keycode*, *XGetKeyboardMapping* returns the symbols for the specified number of keycodes. The specified *first_keycode* must be greater than or equal to *min_keycode* as returned in the *Display* structure, otherwise a *BadValue* error occurs. In addition, the following expression must be less than or equal to *max_keycode* as returned in the *Display* structure, otherwise a *BadValue* error occurs.

first_keycode + *keycode_count* - 1

The number of elements in the keysyms list is:

keycode_count * *keysyms_per_keycode*

Then, *KeySym* number N (counting from zero) for keycode K has an index (counting from zero) of the following (in keysyms):

(K - *first_keycode*) * *keysyms_per_keycode* + N

The *keysyms_per_keycode* value is chosen arbitrarily by the server to be large enough to report all requested symbols. A special *KeySym* value of *NoSymbol* is used to fill in unused elements for individual keycodes.

Use *XFree* to free the returned KeySym list when you no longer need it.

ERRORS

BadValue *first_keycode* less than *display*->**min_keycode**.

display->**max_keycode** exceeded.

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XGetModifierMapping — obtains modifier key mapping (Shift, Control, etc.)

SYNOPSIS

```
XModifierKeymap *XGetModifierMapping(display)
  Display *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XGetModifierMapping returns the keycodes of the keys being used as modifiers.

There are eight modifiers, represented by the symbols *ShiftMapIndex, LockMapIndex, ControlMapIndex, Mod1MapIndex, Mod2MapIndex, Mod3MapIndex, Mod4MapIndex,* and *Mod5MapIndex.* The *modifiermap* member of the *XModifierKeymap* structure contains eight sets of keycodes, each set containing *max_keypermod* keycodes. Zero keycodes are not meaningful. If an entire *modifiermap* is filled with zeroes, the corresponding modifier is disabled. No keycode will appear twice anywhere in the map.

STRUCTURES

```
typedef struct {
  int max_keypermod;    /* server's max number of keys per */
  KeyCode *modifiermap; /* modifier an 8 by max_keypermod array of
                         * keycodes to be used as modifiers */

} XModifierKeymap;

/* modifier names.  Used to build a SetModifierMapping request or
 * to read a GetModifierMapping request.  These correspond to the
 * masks defined above. */

#define ShiftMapIndex         0
#define LockMapIndex          1
#define ControlMapIndex       2
#define Mod1MapIndex          3
#define Mod2MapIndex          4
#define Mod3MapIndex          5
#define Mod4MapIndex          6
#define Mod5MapIndex          7
```

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping.

NAME

XGetMotionEvents — get pointer motion events.

SYNOPSIS

```
XTimeCoord *XGetMotionEvents(display, w, start, stop,
nevents)
  Display *display;
  Window w;
  Time start, stop;
  int *nevents;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose associated pointer motion events will be returned.
start *stop*	Specify the time interval in which the events are returned from the motion history buffer. Pass a time stamp (in milliseconds) or *CurrentTime.*
nevents	Returns the number of events returned from the motion history buffer.

DESCRIPTION

XGetMotionEvents 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. The x and y coordinates of the *XTimeCoord* return structure are reported relative to the origin of *w*.

If the start time is later than the stop time, or if the start time is in the future, no events are returned. If the stop time is in the future, it is equivalent to specifying the constant *CurrentTime.*

The motion history buffer may not be available on all servers. If *display.motion_buffer > 0,* it exists. The pointer position at each pointer hardware interrupt may then be stored for later retrieval.

Use *XFree* to free the returned *XTimeCoord* structures when they are no longer needed.

If *XGetMotionEvents* fails for any reason, it returns NULL.

STRUCTURES

```
typedef struct _XTimeCoord {
  Time time;
  unsigned short x, y;
} XTimeCoord;
```

ERRORS

BadWindow

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XGetNormalHints — get size hints for window in normal state (not zoomed or iconified).

SYNOPSIS

```
Status XGetNormalHints(display, w, hints)
  Display *display;
  Window w;
  XSizeHints *hints;        /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
hints	Returns the sizing hints for the window in its normal state.

DESCRIPTION

XGetNormalHints returns the size hints for a window in its normal state by reading the NORMAL_HINTS property. This function is normally used only by a window manager. It returns a non-zero status if it succeeds, and 0 if it fails (e.g. the application specified no normal size hints for this window.)

STRUCTURES

```
typedef struct {
  long flags;   /* which fields in structure are defined */
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
                             int x;/* numerator */
                             int y;/* denominator */
  } min_aspect, max_aspect;
} XSizeHints;

/* flags argument in size hints */
#define USPosition (1L << 0) /* user specified x, y*/
#define USSize     (1L << 1) /* user specified width, height*/
```

```
#define PPosition  (1L << 2) /* program specified position*/
#define PSize      (1L << 3) /* program specified size*/
#define PMinSize   (1L << 4) /* program specified minimum size*/
#define PMaxSize   (1L << 5) /* program specified maximum size*/
#define PResizeInc (1L << 6) /* program specified resize increments*/
#define PAspect    (1L << 7) /* program specified min/max aspect rat
#define PAllHints (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAsp
```

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGetPixel — obtain a single pixel value from an image.

SYNOPSIS

```
unsigned long XGetPixel(ximage, x, y)
  XImage *ximage;
  int x;
  int y;
```

ARGUMENTS

ximage	Specifies a pointer to the image.
x	
y	Specify the x and y coordinates.

DESCRIPTION

XGetPixel returns the specified pixel from the named image. The *x* and *y* coordinates are relative to the origin (upper left [0,0]) of the image). The pixel value is returned in normalized format; that is, the least significant byte (LSB) of the long is the least significant byte of the pixel.

STRUCTURES

```
typedef struct _XImage {
  int width, height;           /* size of image*/
  int xoffset;                 /* number of pixels offset in X direction*/
  int format;                  /* XYBitmap, XYPixmap, ZPixmap*/
  char *data;                  /* pointer to image data*/
  int byte_order;              /* data byte order, LSBFirst, MSBFirst*/
  int bitmap_unit;             /* quant. of scanline 8, 16, 32*/
  int bitmap_bit_order;        /* LSBFirst, MSBFirst*/
  int bitmap_pad;              /* 8, 16, 32 either XY or ZPixmap*/
  int depth;                   /* depth of image*/
  int bytes_per_line;          /* accelarator to next line*/
  int bits_per_pixel;          /* bits per pixel (ZPixmap)*/
  unsigned long red_mask;      /* bits in z arrangment*/
  unsigned long green_mask;
  unsigned long blue_mask;
  char *obdata;                /* hook for the object routines to hang on */
  struct funcs {               /* image manipulation routines */
  struct _XImage *(*create_image)();
  int (*destroy_image)();
  unsigned long (*get_pixel)();
  int (*put_pixel)();
```

```
        struct _XImage *(*sub_image)();
        int (*add_pixel)();
        } f;
      } XImage;
```

SEE ALSO

XDestroyImage, XPutImage, XGetImage, XCreateImage, XSubImage, XGetSubImage, XAddPixel, XPutPixel, ImageByteOrder.

NAME

XGetPointerControl — get current pointer acceleration parameters.

SYNOPSIS

```
XGetPointerControl(display, accel_numerator, accel_denominator,
threshold)
  Display *display;
  int *accel_numerator, *accel_denominator; /* RETURN */
  int *threshold;
/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
accel_numerator	Returns the numerator for the acceleration multiplier.
accel_denominator	Returns the denominator for the acceleration multiplier.
threshold	Returns the acceleration threshold in pixels. The pointer must move more than this amount before acceleration takes effect.

DESCRIPTION

XGetPointerControl gets the pointer acceleration parameters. *accel_numerator/ accel_denominator* is the number of pixels the cursor moves per unit of motion of the pointer, applied only to the amount of movement over *threshold*.

SEE ALSO

XQueryPointer, XWarpPointer, XGrabPointer, XChangeActivePointerGrab, XUngrabPointer, XGetPointerMapping, XSetPointerMapping, XChangePointerControl.

NAME

XGetPointerMapping — get the pointer button mapping.

SYNOPSIS

```
int XGetPointerMapping(display, map, nmap)
  Display *display;
  unsigned char map[];   /* RETURN */
  int nmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
map	Returns the mapping list. Array begins with **map[]**.
nmap	Specifies the number of items in mapping list.

DESCRIPTION

XGetPointerMapping returns the current mapping of the pointer buttons, in two forms, the args and the returned value. *map* is an array of the numbers of the buttons as they are currently mapped. Elements of the list are indexed starting from one. The nominal mapping for a pointer is the identity mapping: *map[i]=i*. If *map[3]=2*, it means that the third physical button triggers the second logical button.

nmap indicates the desired number of button mappings.

The returned value is the actual number of elements in the pointer list, which may be greater or less than *nmap*.

SEE ALSO

XQueryPointer, XWarpPointer, XGrabPointer, XChangeActivePointerGrab, XUngrabPointer, XSetPointerMapping, XGetPointerControl, XChangePointerControl.

NAME

XGetScreenSaver — get current screen saver parameters.

SYNOPSIS

```
XGetScreenSaver (display, timeout, interval, prefer_blanking,
allow_exposures)
  Display  *display;
  int  *timeout,  *interval;     /*  RETURN  */
  int  *prefer_blanking;         /*  RETURN  */
  int  *allow_exposures;         /*  RETURN  */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
timeout	Returns the timeout, in seconds, until the screen saver turns on.
interval	Returns the interval between screen saver invocations, in seconds.
prefer_blanking	Returns the current screen blanking preference, one of these constants: *DontPreferBlanking, PreferBlanking,* or *DefaultBlanking*.
allow_exposures	Returns the current screen save control value, either *DontAllowExposures, AllowExposures,* or *DefaultExposures*.

DESCRIPTION

XGetScreenSaver returns the current settings of the screen saver, which may be set with *XSetScreenSaver.*

A positive *timeout* indicates that the screen saver is enabled. A *timeout* of *0* indicates that the screen saver is disabled. If no input from devices (keyboard, mouse, etc.) is generated for the specified number of *timeout* seconds, the screen saver is activated.

If the server-dependent screen saver method supports periodic change, *interval* serves as a hint about how long the change period is, and zero hints that no periodic change should be made. Examples of ways to change the screen include scrambling the color map periodically, moving an icon image about the screen periodically, or tiling the screen with the root window background tile, randomly reoriginated periodically. An *interval* of *0* indicates that random pattern motion is disabled.

SEE ALSO

XForceScreenSaver, XActivateScreenSaver, XResetScreenSaver, XSetScreenSaver.

NAME

XGetSelectionOwner — return selection owner.

SYNOPSIS

Window XGetSelectionOwner(*display*, *selection*)
  Display **display*;
  Atom *selection*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
selection	Specifies the selection atom. This is the atom whose owner you want returned.

DESCRIPTION

XGetSelectionOwner returns the window ID of the current owner of the specified selection. If no selection was specified, or there is no owner, the function returns the constant *None*.

ERRORS

BadAtom

SEE ALSO

XSetSelectionOwner, *XConvertSelection*.

NAME

XGetSizeHints — read any property of type *WM_SIZE_HINTS*.

SYNOPSIS

```
Status XGetSizeHints(display, w, hints, property)
  Display *display;
  Window w;
  XSizeHints *hints;        /* RETURN */
  Atom property;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
hints	Returns the size hints structure.
property	Specifies the property atom.

DESCRIPTION

XGetSizeHints returns the *XSizeHints* structure for the named property and the specified window. This is used by *XGetNormalHints* and *XGetZoomHints*, and can be used to retrieve the value of any property of type *WM_SIZE_HINTS*; thus, it is useful if other properties of that type get defined. These functions are used almost exclusively by window managers.

XGetSizeHints returns a non-zero status if a size hint was defined, or returns 0 otherwise.

STRUCTURES

```
typedef struct {
  long flags;              /* which fields in structure are defined
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
          int x;                         /* numerator */
          int y;                         /* denominator */
  } min_aspect, max_aspect;
} XSizeHints;
```

```
/* flags argument in size hints */
#define USPosition (1L << 0)  /* user specified x, y */
#define USSize     (1L << 1)  /* user specified width, height */

#define PPosition  (1L << 2)  /* program specified position */
#define PSize      (1L << 3)  /* program specified size */
#define PMinSize   (1L << 4)  /* program specified minimum size */
#define PMaxSize   (1L << 5)  /* program specified maximum size */
#define PResizeInc (1L << 6)  /* program specified resize increments */
#define PAspect    (1L << 7)  /* program specified min/max aspect ratios */
#define PAllHints  (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
```

ERRORS

BadAtom
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGetStandardColormap — get standard colormap structure.

SYNOPSIS

```
Status XGetStandardColormap(display, w, cmap, property)
  Display *display;
  Window w;
  XStandardColormap *cmap;/* RETURN */
  Atom property;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
cmap	Returns the filled colormap information structure.
property	Specifies the atom indicating the type of standard colormap desired. The pre-defined standard colormap atoms are *XA_RGB_BEST_MAP*, *XA_RGB_RED_MAP*, *XA_RGB_GREEN_MAP*, *XA_RGB_BLUE_MAP*, *XA_DEFAULT_MAP*, and *XA_RGB_GRAY_MAP*.

DESCRIPTION

The *XGetStandardColormap* function returns the colormap definition associated with the atom supplied as the property argument. For example, to fetch the standard gray-scale colormap for a display, you use *XGetStandardColormap* with the following syntax.

```
XStandardColormap colormap;

XGetStandardColormap(dpy, RootWindow(dpy, 0), &colormap, XA_RGB_GRAY_MAP);
```

This call does not load the colormap into the hardware colormap, it does not allocate entries, and it does not even create a virtual colormap. It just provides information about one colormap. The application can then attempt to create a virtual colormap of the appropriate type, and allocate its entries according to the information in the *XStandardColormap*. Installing the standard colormap must then be done with *XInstallColormap*, in cooperation with the window manager. Any of these steps could fail, and the application should be prepared.

An application should go through this process only if it needs the special qualities of the standard colormaps. For one, they allow you to convert

RGB values into pixel values easily. Given an *XStandardColormap* structure for a *XA_RGB_BEST_MAP* colormap, and floating point RGB coefficients in the range 0.0 to 1.0, you can compose pixel values with the following C expression:

```
pixel = base_pixel
  + ((unsigned long) (0.5 + r * red_max)) * red_mult
  + ((unsigned long) (0.5 + g * green_max)) * green_mult
  + ((unsigned long) (0.5 + b * blue_max)) * blue_mult;
```

The use of addition rather than logical-OR for composing pixel values permits allocations where the RGB value is not aligned to bit boundaries.

Refer to the *GSE Programmer's Guide* for a complete description of standard colormaps.

STRUCTURES

```
typedef struct {
  Colormap colormap;     /* ID of colormap created by XCreateColormap */
  unsigned long red_max;
  unsigned long red_mult;
  unsigned long green_max;
  unsigned long green_mult;
  unsigned long blue_max;
  unsigned long blue_mult;
  unsigned long base_pixel;
} XStandardColormap;
```

ERRORS

BadAtom
BadWindow

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XFreeColormap, XInstallColormap, XUninstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XGetSubImage — copy rectangle in drawable to location within pre-existing image.

SYNOPSIS

```
XImage *XGetSubImage(display, d, x, y, width, height,
plane_mask, format, dest_image, dest_x, dest_y)
  Display *display;
  Drawable d;
  int x, y;
  unsigned int width, height;
  unsigned long plane_mask;
  int format;
  XImage *dest_image;
  int dest_x, dest_y;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable, a pixmap, or window.
x *y*	Specify the x and y coordinates. These coordinates define the upper left corner of the rectangle relative to the origin of the drawable.
width *height*	Specify the width and height of the subimage taken.
plane_mask	Specifies which planes of the drawable are transferred to image.
format	Specifies the format for the image. Either *XYPixmap* or *ZPixmap*.
dest_image	Specifies the the destination image.
dest_x *dest_y*	Specify the x and y coordinates of the destination rectangle relative to the image's origin. They specify the upper left corner of the destination rectangle in the image, determining where the subimage will be placed.

DESCRIPTION

XGetSubImage updates the *dest_image* with the specified subimage in the same manner as *XGetImage,* except that it does not create the image or necessarily fill the entire image. If *format* is *XYPixmap,* the function transmits only the bit planes you specify in *plane_mask.* If *format* is *ZPixmap,* the function transmits as zero the bits in all planes not specified in *plane_mask.* The function performs no range checking on the values in *plane_mask* and ignores extraneous bits.

The depth of the destination *XImage* structure must be the same as that of the drawable. Otherwise, a *BadImage* error is generated. If the specified subimage does not fit at the specified location on the destination image, the right and bottom edges are clipped. If the drawable is a window, the window must be mapped or held in backing store. It must also be the case that, if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen; otherwise, a *BadMatch* error is generated.

If the window has a backing-store, the backing-store contents are returned for regions of the window that are obscured by noninferior windows. Otherwise, the return contents of such obscured regions are undefined. Also undefined are the returned contents of visible regions of inferiors of different depth than the specified window.

XSubImage extracts a subimage from an image, instead of from a drawable like *XGetSubImage.*

ERRORS

BadDrawable	
BadGC	
BadMatch	Depth of *dest_image* is not the same as depth of *d*. See also Description.
BadValue	

SEE ALSO

XDestroyImage, XPutImage, XGetImage, XCreateImage, XSubImage, XAddPixel, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XGetTransientForHint — get WM_TRANSIENT_FOR property of window.

SYNOPSIS

```
Status XGetTransientForHint(display, w, prop_window)
  Display *display;
  Window w;
  Window *prop_window;       /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
prop_window	Returns the WM_TRANSIENT_FOR property of the specified window.

DESCRIPTION

XGetTransientForHint obtains the WM_TRANSIENT_FOR property for the specified window. This function is normally used by a window manager. This property should be set for windows that are to appear only temporarily on the screen, such as pop-up menus and dialog boxes.

XGetTransientForHint returns a status of 0 on failure, non-zero on success.

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGetVisualInfo — find visual information structure that matches template.

SYNOPSIS

```
XVisualInfo *XGetVisualInfo(display, vinfo_mask,
vinfo_template, nitems)
  Display *display;
  long vinfo_mask;
  XVisualInfo *vinfo_template;
  int *nitems;                /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
vinfo_mask	Specifies the visual mask value. Indicates which elements in template are to be matched.
vinfo_template	Specifies the visual attributes that are to be used in matching the visual structures.
nitems	Returns the number of matching visual structures.

DESCRIPTION

XGetVisualInfo returns a list of visual structures that match the attributes specified by the *vinfo_template* argument. If no visual structures match the template, *XGetVisualInfo* returns a *NULL*. To free the data returned by this function, use *XFree*.

STRUCTURES

```
typedef struct {
  Visual *visual;
  VisualID visualid;
  int screen;
  unsigned int depth;
  int class;
  unsigned long red_mask;
  unsigned long green_mask;
  unsigned long blue_mask;
  int colormap_size;
  int bits_per_rgb;
} XVisualInfo;
```

```
/* The symbols for the vinfo_mask argument are: */

#define VisualNoMask            0x0
#define VisualIDMask            0x1
#define VisualScreenMask        0x2
#define VisualDepthMask         0x4
#define VisualClassMask         0x8
#define VisualRedMaskMask       0x10
#define VisualGreenMaskMask     0x20
#define VisualBlueMaskMask      0x40
#define VisualColormapSizeMask 0x80
#define VisualBitsPerRGBMask    0x100
#define VisualAllMask           0x1FF
```

SEE ALSO

XMatchVisualInfo, DefaultVisual.

NAME

XGetWindowAttributes — obtain current attributes of window.

SYNOPSIS

```
Status XGetWindowAttributes(display, w, window_attributes)
  Display *display;
  Window w;
  XWindowAttributes *window_attributes;/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window whose current attributes you want.
window_attributes	Returns a filled *XWindowAttributes* structure, containing the current attributes for the specified window.

DESCRIPTION

XGetWindowAttributes returns the *XWindowAttributes* structure containing the current window attributes.

While *w* is defined as type *Window*, a *Pixmap* can also be used, in which case all the returned members will be zero except *width, height, depth,* and *screen.*

The following list briefly describes each member. For more information, refer to the *GSE Programmer's Guide.*

`x, y`	The current position of this window relative to its parent.
`width, height`	The current dimensions of this window.
`depth`	The number of bits per pixel in this window.
`visual`	The visual structure.
`root`	The root window ID of the screen containing the window.
`class`	The window class. One of these constants: *InputOutput* or *InputOnly*.

bit_gravity	The new position for existing contexts on resize. One of the constants *ForgetGravity*, *StaticGravity*, or *CenterGravity*, or one of the compass constants (*NorthWestGravity*, *NorthGravity*, etc.).
win_gravity	The new position for subwindow on parent resize. One of the constants *CenterGravity*, *UnmapGravity*, *StaticGravity*, or one of the compass constants.
backing_store	When to maintain contents of the window. One of these constants: *NotUseful*, *WhenMapped*, or *Always*.
backing_planes	The bit planes to be preserved in a backing store.
backing_pixel	The pixel value used when restoring planes from a partial backing store.
save_under	A boolean value, indicating whether saving bits under this window would be useful.
colormap	The colormap ID to be used in this window, or *None*.
map_installed	A boolean value, indicating whether the colormap is currently installed. If *True*, the window is being displayed in its chosen colors.
map_state	The window's map state. One of these constants: *IsUnmapped*, *IsUnviewable*, or *IsViewable*. *IsUnviewable* indicates that the specified window is mapped but some ancestor is unmapped.
all_event_masks	The set of events any client have selected. This member is the bitwise inclusive OR of all event masks selected on the window by all clients.

`your_event_mask`
: The bitwise inclusive OR of all event mask symbols selected by the querying client.

`do_not_propagate_mask`
: The bitwise inclusive OR of the event mask symbols that specify the set of events that should not propagate. This is global across all clients.

`override_redirect`
: A boolean value, indicating whether this window will override structure control facilities. This is usually only used for temporary pop-up windows. Either *True* or *False*.

`screen`
: A pointer to the *Screen* structure for the screen containing this window.

XGetWindowAttributes returns a status of 0 on failure or 1 when it succeeds.

STRUCTURES

The XWindowAttributes structure contains:

```
typedef struct {
  int x, y;                       /* location of window*/
  int width, height;              /* width and height of window*/
  int border_width;               /* border width of window*/
  int depth;                      /* depth of window*/
  Visual *visual;                 /* the associated visual structure*/
  Window root;                    /* root of screen containing window*/
  int class;                      /* InputOutput, InputOnly*/
  int bit_gravity;                /* one of bit gravity values*/
  int win_gravity;                /* one of the window gravity values*/
  int backing_store;              /* NotUseful, WhenMapped, Always*/
  unsigned long backing_planes;/* planes to be preserved if possible*/
  unsigned long backing_pixel; /* value to be used when restoring planes*/
  Bool save_under;                /* boolean, should bits under be saved*/
  Colormap colormap;              /* colormap to be associated with window*/
  Bool map_installed;             /* boolean, is colormap currently installed*/
  int map_state;                  /* IsUnmapped, IsUnviewable, IsViewable*/
  long all_event_masks;           /* set of events all people have interest in*/
  long your_event_mask;           /* my event mask*/
  long do_not_propagate_mask;     /* set of events that should not propagate*/
```

```
    Bool override_redirect;      /* boolean value for override-redirect
    Screen *screen;              /* pointer to correct screen*/
  } XWindowAttributes;
```

SEE ALSO

XChangeWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap, XGetGeometry.

NAME

XGetWindowProperty — obtain the atom type and property format for a window.

SYNOPSIS

```
int XGetWindowProperty(display, w, property, long_offset,
long_length, delete, req_type, actual_type, actual_format, nitems,
bytes_after, prop)
  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 */
  long *bytes_after;          /* RETURN */
  unsigned char **prop;       /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose atom type and property format you want to obtain.
property	Specifies the property atom.
long_offset	Specifies the offset in 32-bit quantities where data will be retrieved.
long_length	Specifies the length in 32-bit multiples of the data to be retrieved.
delete	Specifies a boolean value of *True* or *False*. If you pass *True* and a property is returned, the property is deleted from the window and a *PropertyNotify* event is generated on the window.

req_type	If *AnyPropertyType* is specified, returns the property from the specified window regardless of its type. If a type is specified, the function returns the property only if its type equals the specified type.
actual_type	Returns the actual type of the property.
actual_format	Returns the actual data type of the returned data.
nitems	Returns the actual number of 8-, 16-, or 32-bit items returned in *prop*.
bytes_after	Returns the number of bytes remaining to be read in the property if a partial read was performed.
prop	Returns a pointer to the data actually returned, in the specified format. *XGetWindowProperty* always allocates one extra byte after the data and sets it to ASCII Null. This byte is not counted in *nitems*.

DESCRIPTION

XGetWindowProperty gets the value of a property if it is the desired type. *XGetWindowProperty* sets the return arguments acccording to the following rules:

- If the specified property does not exist for the specified window, *actual_type* is *None, actual_format* = 0 and *bytes_after* = 0. *delete* is ignored in this case, and *nitems* is empty.
- If the specified property exists, but its type does not match *req_type, actual_type* is the actual property type, *actual_format;* is the actual property format (never zero), and *bytes_after* is the property length in bytes (even if *actual_format* is 16 or 32). *delete* is ignored in this case, and *nitems* is empty.
- If the specified property exists, and either *req_type* is *AnyPropertyType* or the specified type matches the actual property type, *actual_type* is the actual property type and *actual_format* is the actual property format (never zero). *bytes_after* and *nitems* are defined by combining the following values:

```
N = actual length of stored property in bytes (even if actual_format is 16 or 32)
I = 4 * long_offset (convert offset from longs into bytes)
L = MINIMUM((N - I), 4 * long_length) (BadValue if L < 0)
bytes_after = N - (I + L) (number of trailing unread bytes in stored property
```

The returned data (in *prop*) starts at byte index I in the property (indexing from 0). The actual length of the returned data in bytes is *L*. *L* is converted into the number of 8-, 16-, or 32-bit items returned by dividing by 1, 2, or 4 respectively and this value is returned in *nitems*. The number of trailing unread bytes is returned in *bytes_after*.

If *delete* == `True` and *bytes_after* == `0` the function deletes the property from the window and generates a *PropertyNotify* event on the window.

RETURNED VALUE

When *XGetWindowProperty* executes successfully, it returns *Success*. If the specified window did not exist, it generates a *BadWindow* error. If the type you passed in *req_type* did not exist or did not match the property type returned in *actual_type*, the function generates a *BadMatch* error.

ERRORS

BadValue	Value of *long_offset* caused *L* to be negative above.
BadAtom	
BadWindow	
BadMatch	

SEE ALSO

XSetStandardProperties, XGetFontProperty, XRotateWindowProperties, XDeleteProperty, XChangeProperty, XListProperties, XGetAtomName, XInternAtom.

NAME

XGetWMHints — read window manager hints.

SYNOPSIS

```
XWMHints  *XGetWMHints(display, w)
  Display  *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

This function is primarily for window managers. *XGetWMHints* returns *NULL* if no WM_HINTS property was set on window *w,* and returns a pointer to a *XWMHints* structure if it succeeds. Programs must free the space used for that structure by calling *XFree.*

STRUCTURES

```
typedef struct {
  long flags;          /* marks which fields in this structure are defined*/
  Bool input;          /* does application need window manager for input*/
  int initial_state;   /* see below*/
  Pixmap icon_pixmap;  /* pixmap to be used as icon*/
  Window icon_window;  /* window to be used as icon*/
  int icon_x, icon_y;  /* initial position of icon*/
  Pixmap icon_mask;    /* icon mask bitmap*/
  XID window_group;    /* id of related window group*/

  /* this structure may be extended in the future */

} XWMHints;

/* initial state flag: */
#define DontCareState        0
#define NormalState          1
#define ZoomState            2
#define IconicState          3
#define InactiveState        4
```

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGetZoomHints — read size hints for zoomed window.

SYNOPSIS

```
Status XGetZoomHints(display, w, zhints)
  Display *display;
  Window w;
  XSizeHints *zhints;        /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
zhints	Returns a pointer to the zoom hints.

DESCRIPTION

XGetZoomHints is primarily for window managers. *XGetZoomHints* returns the size hints for a window in its zoomed state (not normal or iconified) read from the WM_ZOOM_HINTS property. It returns a non-zero status if it succeeds, and 0 if the application did not specify zoom size hints for this window.

STRUCTURES

```
typedef struct {
  long flags;     /* which fields in structure are defined */
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
          int x; /* numerator */
          int y; /* denominator */
  } min_aspect, max_aspect;
} XSizeHints;

/* flags argument in size hints */
#define USPosition (1L << 0)  /* user specified x, y*/
#define USSize     (1L << 1)  /* user specified width, height*/

#define PPosition  (1L << 2)  /* program specified position*/
#define PSize      (1L << 3)  /* program specified size*/
```

```
#define PMinSize    (1L << 4)  /* program specified minimum size*/
#define PMaxSize    (1L << 5)  /* program specified maximum size*/
#define PResizeInc  (1L << 6)  /* program specified resize increments*/
#define PAspect     (1L << 7)  /* program specified min/max aspect ratios*/
#define PAllHints   (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
```

ERRORS

BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XGrabButton — grab a pointer button.

SYNOPSIS

```
XGrabButton(display, button, modifiers, grab_window, owner_events,
event_mask, pointer_mode, keyboard_mode, confine_to, cursor)
  Display *display;
  unsigned int button;
  unsigned int modifiers;
  Window grab_window;
  Bool owner_events;
  unsigned int event_mask;
  int pointer_mode, keyboard_mode;
  Window confine_to;
  Cursor cursor;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
button	Specifies the mouse button. May be *Button1, Button2, Button3, Button4, Button5,* or *AnyButton.* The constant *AnyButton* is equivalent to issuing the grab request for all possible buttons. The button symbols cannot be ORed.
modifiers	Specifies a set of keymasks. This is a bitwise OR of one or more of the following symbols: *ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask,* or *AnyModifier. AnyModifier* is equivalent to issuing the grab key request for all possible modifier combinations (including no modifiers).
grab_window	Specifies the window ID. This is the window you want to grab.
owner_events	Specifies a boolean value of either *True* or *False.* See Description below.
event_mask	Specifies the event mask. This mask is the bitwise OR of one or more of the event masks listed under *XSelectInput.*

pointer_mode	Controls further processing of pointer events. Pass one of these constants: *GrabModeSync* or *GrabModeAsync*.
keyboard_mode	Controls further processing of keyboard events. Pass one of these constants: *GrabModeSync* or *GrabModeAsync*.
confine_to	Specifies the window to confine the pointer. One possible value is the constant *None*, in which case the pointer is not confined to any window.
cursor	Specifies the cursor to be displayed during the grab. One possible value you can pass is the constant *None*.

DESCRIPTION

XGrabButton establishes a passive grab, such that an active grab may take place when the specified key/button combination is pressed. After this call, if

1) the specified button is pressed when the specified modifier keys are down (and no other buttons or modifier keys are down),
2) *grab_window* contains the pointer,
3) the *confine_to* window (if any) is viewable, and
4) these constraints are not satisfied for any ancestor,

then the pointer is actively grabbed as described in *GrabPointer*, the *last_pointer_grab* time is set to the time at which the button was pressed, and the *ButtonPress* event is reported.

The interpretation of the remaining arguments is as for *XGrabPointer*. The active grab is terminated automatically when all buttons are released (independent of the state of modifier keys).

A modifier of *AnyModifier* is equivalent to issuing the grab request for all possible modifier combinations (including no modifiers). A button of *AnyButton* is equivalent to issuing the request for all possible buttons (but at least one).

The request fails if some other client has already issued a *GrabButton* with the same button/key combination on the same window. When using *AnyModifier* or *AnyButton*, the request fails completely (no grabs are established) if there is a conflicting grab for any combination. The request has no effect on an active grab.

The *owner_events* argument specifies whether the grab window should receive all events (*True*) or whether the grabbing application should receive all events normally (*False*).

The *pointer_mode* and *keyboard_mode* control the processing of events during the grab. If either is *GrabModeSync,* events for that device are not queued for applications until *XAllowEvents* is called to release the events. If either is *GrabModeAsync,* events for that device are processed normally.

An automatic grab takes place between a *ButtonPress* and a *ButtonRelease,* so this call is not necessary in some of the most common situations. Refer to the description of grabbing in the *GSE Programmer's Guide.*

ERRORS

BadAccess	When using *AnyModifier* or *AnyButton* and there is a conflicting grab by another client. No grabs are established.
--	Another client has already issued an *XGrabButton* request with the same key/button combination on the same window.
BadAlloc	
BadCursor	
BadValue	
BadWindow	

SEE ALSO

XGrabKey, XUngrabKey, XGrabKeyboard, XUngrabKeyboard, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XGrabKey — grab a key.

SYNOPSIS

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*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
keycode	Specifies the keycode to be grabbed. It may be a modifier key. Specifying *AnyKey* is equivalent to issuing the request for all key codes.
modifiers	Specifies a set of keymasks. This is a bitwise OR of one or more of the following symbols: *ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask,* or *AnyModifier*. *AnyModifier* is equivalent to issuing the grab key request for all possible modifier combinations (including no modifiers). All specified modifiers do not need to have currently assigned keycodes.
grab_window	Specifies the window from which you want to receive input from the grabbed key combination.
owner_events	Specifies whether the grab window should receive all events (*True*) or whether the grabbing application should receive all events normally (*False*).
pointer_mode	Controls further processing of pointer events. Pass one of these constants: *GrabModeSync* or *GrabModeAsync*.
keyboard_mode	Controls further processing of keyboard events. Pass one of these constants: *GrabModeSync* or *GrabModeAsync*.

DESCRIPTION

XGrabKey establishes a passive grab on the specified keys, such that when the specified key/modifier combination is pressed, the keyboard is grabbed, and all keyboard events are sent to this application. More formally:

- IF the keyboard is not grabbed and the specified key, which itself can be a modifier key, is logically pressed when the specified modifier keys logically are down (and no other keys are down),
- AND no other modifier keys logically are down,
- AND EITHER the grab window is an ancestor of (or is) the focus window OR the grab window is a descendent of the focus window and contains the pointer,
- AND a passive grab on the same key combination does not exist on any ancestor of the grab window,
- THEN the keyboard is actively grabbed, as for *XGrabKeyboard,* the last keyboard grab time is set to the time at which the key was pressed (as transmitted in the *KeyPress* event), and the *KeyPress* event is reported.

The active grab is terminated automatically when the specified key is released (independent of the state of the modifier keys).

The *pointer_mode* and *keyboard_mode* control the processing of events during the grab. If either is *GrabModeSync,* events for that device are not queued for applications until *XAllowEvents* is called to release the events. If either is *GrabModeAsync,* events for that device are processed normally.

ERRORS

BadAccess --	When using *AnyModifier* or *AnyKey* and another client has grabbed any overlapping combinations. In this case, no grabs are established.
--	Another client has issued *XGrabKey* for the same key combination in *grab_window.*
BadValue	*keycode* is not in the range between *min_keycode* and *max_keycode* in the display structure.
BadWindow	

SEE ALSO

XUngrabKey, XGrabKeyboard, XUngrabKeyboard, XGrabButton, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XGrabKeyboard — grab the keyboard.

SYNOPSIS

```
int XGrabKeyboard(display, w, owner_events, pointer_mode,
keyboard_mode, time)
  Display *display;
  Window grab_window;
  Bool owner_events;
  int pointer_mode, keyboard_mode;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
grab_window	Specifies the window that requires continuous keyboard input.
owner_events	Specifies a boolean value of either *True* or *False*. See Description below.
pointer_mode	Controls further processing of pointer events. Pass either *GrabModeSync* or *GrabModeAsync*.
keyboard_mode	Controls further processing of keyboard events. Pass either *GrabModeSync* or *GrabModeAsync*.
time	Specifies the time when the grab should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime*.

DESCRIPTION

XGrabKeyboard actively grabs control of the main keyboard. If the grab is successful, it returns the constant *GrabSuccess*. Further key events are reported only to the grabbing client. This request generates *FocusIn* and *FocusOut* events.

XGrabKeyboard processing is controlled by the value in the *owner_events* argument:

- If *owner_events* is *False*, all generated key events are reported to *grab_window*.

- If *owner_events* is *True*, then if a generated key event would normally be reported to this client, it is reported normally. Otherwise the event is reported to *grab_window*.

Both *KeyPress* and *KeyRelease* events are always reported, independent of any event selection made by the client.

XGrabKeyboard processing of pointer events and keyboard events are controlled by *pointer_mode* and *keyboard_mode*:

- If the *pointer_mode* or *keyboard_mode* is *GrabModeAsync*, event processing for the respective device continues normally.
- For *keyboard_mode GrabModeAsync* only: if the keyboard was currently frozen by this client, then processing of keyboard events is resumed.
- If the *pointer_mode* or *keyboard_mode* is *GrabModeSync*, events for the respective device are queued until a releasing *XAllowEvents* request or until the keyboard grab is released as described above.

XGrabKeyboard processing fails under the following conditions and returns the following:

- If the keyboard is actively grabbed by some other client, it returns *AlreadyGrabbed*.
- If *grab_window* is not viewable, it returns *GrabNotViewable*.
- If *time* is earlier than the last keyboard grab time or later than the current server time, it returns *GrabInvalidTime*.
- If the pointer is frozen by an active grab of another client, the request fails with a status *GrabFrozen*.

If the grab succeeds, the last keyboard grab time is set to the specified time, with *CurrentTime* replaced by the current X server time.

ERRORS

BadValue
BadWindow

SEE ALSO

XGrabKey, XUngrabKey, XUngrabKeyboard, XGrabButton, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XGrabPointer — grab the pointer.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
grab_window	Specifies the window that wants to grab the pointer input independent of pointer location.
owner_events	Specifies if the pointer events are to be reported normally within this application (pass *True*) or only to the grab window if selected by the event mask (pass *False*).
event_mask	Specifies the event mask. See *XSelectInput* for a complete list of event masks.
pointer_mode	Controls further processing of pointer events. Pass either *GrabModeSync* or *GrabModeAsync*.
keyboard_mode	Controls further processing of keyboard events. Pass either *GrabModeSync* or *GrabModeAsync*.
confine_to	Specifies the window to confine the pointer. One option is *None,* in which case the pointer is not confined to any window.
cursor	Specifies the cursor. This is the cursor that is displayed with the pointer during the grab. One option is *None,* which causes the cursor to keep its current pattern.

time	Specifies the time when the grab request took place. Pass either a timestamp, expressed in milliseconds (from an event), or the constant *CurrentTime*.

DESCRIPTION

XGrabPointer actively grabs control of the pointer. If the grab is successful, it returns the constant *GrabSuccess*. Further pointer events are only reported to the grabbing client.

event_mask is always augmented to include *ButtonPressMask* and *ButtonReleaseMask*. If *owner_events* is *False*, all generated pointer events are reported with respect to *grab_window*, and are only reported if selected by *event_mask*. If *owner_events* is *True*, then if a generated pointer event would normally be reported to this client, it is reported normally; otherwise the event is reported with respect to the *grab_window*, and is only reported if selected by *event_mask*. For either value of *owner_events*, unreported events are discarded.

pointer_mode controls further processing of pointer events, and *keyboard_mode* controls further processing of main keyboard events. If the mode is *GrabModeAsync*, event processing continues normally. If the mode is *GrabModeSync*, events for the device are queued but not sent to clients until the grabbing client issues a releasing *XAllowEvents* request or an *XUngrabPointer* request.

If a cursor is specified, then it is displayed regardless of what window the pointer is in. If no cursor is specified, then when the pointer is in *grab_window* or one of its subwindows, the normal cursor for that window is displayed. Otherwise, the cursor for *grab_window* is displayed.

If a *confine_to* window is specified, then the pointer will be restricted to stay contained in that window. The *confine_to* window need have no relationship to the *grab_window*. If the pointer is not initially in the *confine_to* window, then it is warped automatically to the closest edge (and enter/leave events generated normally) just before the grab activates. If the *confine_to* window is subsequently reconfigured, the pointer will be warped automatically as necessary to keep it contained in the window.

The *time* argument lets you avoid certain circumstances that come up if applications take a long while to respond or if there are long network delays. Consider a situation where you have two applications, both of which normally grab the pointer when clicked on. If both applications specify the timestamp from the *ButtonPress* event, the second application will successfully grab the pointer, while the first will get a return value of

AlreadyGrabbed, indicating that the other application grabbed the pointer before its request was processed. This is the desired response because the latest user actions is most important in this case.

XGrabPointer may generate more than one *EnterNotify* and *LeaveNotify* event pair.

The *XGrabPointer* function fails under the following conditions, with the following return values:

- If *grab_window* or *confine_to* window is not viewable, *GrabNotViewable* is returned.
- If the pointer is actively grabbed by some other client, the constant *AlreadyGrabbed* is returned.
- If the pointer is frozen by an active grab of another client, *GrabFrozen* is returned.
- If the specified time is earlier than the last-pointer-grab time or later than the current X server time, *GrabInvalidTime* is returned. (If the call succeeds, the last pointer grab time is set to the specified time, with the constant *CurrentTime* replaced by the current X server time.)

ERRORS

BadCursor
BadValue
BadWindow

SEE ALSO

XGrabKey, XUngrabKey, XGrabKeyboard, XUngrabKeyboard, XGrabButton, XUngrabButton, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XGrabServer — grab the server.

SYNOPSIS

```
XGrabServer(display)
  Display *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

Grabbing the server means that only requests by the calling client will be acted on. All others will be queued in the server until the next *XUngrabServer* call. The X server should not be grabbed any more than is absolutely necessary.

SEE ALSO

XGrabKey, XUngrabKey, XGrabKeyboard, XUngrabKeyboard, XGrabButton, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XUngrabServer.

NAME

XIfEvent — wait for matching event.

SYNOPSIS

```
XIfEvent(display, event, predicate, args)
  Display *display;
  XEvent *event;               /* RETURN */
  Bool (*predicate)();
  char *args;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event	Returns the matched event.
predicate	Specifies the procedure to be called to determine if the next event satisfies your criteria.
args	Specifies the user-specified arguments to be passed to the predicate procedure.

DESCRIPTION

XIfEvent checks the event queue for events, uses the user-supplied routine to check if they meet certain criteria, and removes the matching event from the input queue. *XIfEvent* returns only when the specified predicate procedure returns *True* for an event. The specified predicate is called each time an event is added to the queue.

If no matching events exist on the queue, *XIfEvent* flushes the output buffer and waits for an appropriate event to arrive. Use *XCheckIfEvent* if you don't want to wait for an event.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XInsertModifiermapEntry — add a new entry to an **XModifierKeymap** structure.

SYNOPSIS

```
XModifierKeymap        *XInsertModifiermapEntry(modmap,
keysym_entry, modifier)
        XModifierKeymap *modmap;
        KeyCode keysym_entry;
        int modifier;
```

ARGUMENTS

modmap	Specifies a pointer to an *XModifierKeymap* structure.
keysym_entry	Specifies the *KeyCode* of the key to be added to *modmap*.
modifier	Specifies the modifier you want mapped to the keycode specified in *keysym_entry*. This should be one of the constants: *ShiftMapIndex, LockMapIndex, ControlMapIndex, Mod1MapIndex, Mod2MapIndex, Mod3MapIndex, Mod4MapIndex,* or *Mod5MapIndex*.

DESCRIPTION

XInsertModifiermapEntry returns an *XModifierKeymap* structure suitable for calling *XSetModifierMapping,* in which the specified keycode is deleted from the set of keycodes that is mapped to the specified modifier (like Shift or Control). *XInsertModifiermapEntry* does not change the mapping itself.

This function is normally used by calling *XGetModifierMapping* to get a pointer to the current *XModifierKeymap* structure for use as the *modmap* argument to *XInsertModifiermapEntry*.

Note that the structure pointed to by *modmap* is freed by *XInsertModifiermapEntry*. It should not be freed or otherwise used by applications.

For a description of the modifier map, see *XSetModifierMapping*.

STRUCTURES

```
typedef struct {
   int max_keypermod;     /* server's max number of keys per modifier
   KeyCode *modifiermap; /* an 8 by max_keypermod array of
                          * keycodes to be used as modifiers */
} XModifierKeymap;


#define ShiftMapIndex     0
```

```
#define LockMapIndex       1
#define ControlMapIndex    2
#define Mod1MapIndex       3
#define Mod2MapIndex       4
#define Mod3MapIndex       5
#define Mod4MapIndex       6
#define Mod5MapIndex       7
```

SEE ALSO

XDeleteModifiermapEntry, XGetModifierMapping, XSetModifierMapping, XNewModifierMap, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XRefreshKeyboardMapping, XLookupString.

NAME

XInstallColormap — install a colormap.

SYNOPSIS

```
XInstallColormap(display, cmap)
  Display *display;
  Colormap cmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap to install.

DESCRIPTION

If there is only one hardware colormap, *XInstallColormap* loads a virtual colormap into the hardware colormap. All windows associated with this colormap immediately display with their chosen colors. Other windows associated with the old colormap will display with false colors.

If additional hardware colormaps are possible, *XInstallColormap* loads the new hardware map and keeps the existing ones. Other windows will then remain in their true colors unless the limit for colormaps has been reached. If the maximum number of allowed hardware colormaps is already installed, an old colormap is swapped out. The *MinCmapsOfScreen(screen)* and *MaxCmapsOfScreen(screen)* macros can be used to determine how many hardware colormaps are supported.

If *cmap* is not already an installed map, a *ColormapNotify* event is generated on every window having *cmap* as an attribute. If a colormap is uninstalled as a result of the install, a *ColormapNotify* event is generated on every window having that colormap as an attribute.

Colormaps are usually installed and uninstalled by the window manager, not by clients.

At any time, there is a subset of the installed colormaps, viewed as an ordered list, called the "required list." The length of the required list is at most the *min_maps* specified for each screen in the *Display* structure. When a colormap is installed with *XInstallColormap* it is added to the head of the required list and the last colormap in the list is removed if necessary to keep the length of the list at *mim_maps*. When a colormap is uninstalled with *XUninstallColormap* and it is in the required list, it is removed from the list. No other actions by the server or the client change the required list. It is important to realize that on all but high-end workstations, *min_maps* is likely to be 1.

ERRORS

BadColor

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XFreeColormap, XGetStandardColormap, XUninstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XInternAtom — return an atom for a name string.

SYNOPSIS

```
Atom XInternAtom(display, atom_name, only_if_exists)
  Display *display;
  char *atom_name;
  Bool only_if_exists;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
atom_name	Specifies the name associated with the atom you want returned. The string should use the ISO Latin-1 encoding, and upper/lower case is important.
only_if_exist	Specifies a boolean value that indicates whether *XInternAtom* should return *None* or should create the atom if no such *atom_name* exists.

DESCRIPTION

If the atom exists, *XInternAtom* returns the atom identifier corresponding to *atom_name*.

If the atom does not exist, then *XInternAtom* either returns *None* (if *only_if_exists* is *True*) or creates the atom (if *only_if_exists* is *False*). The string name should be a null-terminated ASCII string. Case matters; the strings "thing", "Thing", and "thinG" all designate different atoms. The atom remains defined even after the client who defined it has exited. It becomes undefined only when the last connection to the X server closes.

This function is the opposite of *XGetAtomName,* which returns the atom name when given an atom ID.

Predefined atoms are defined in <*X11/Xatom.h*> and begin with the prefix "XA_".

ERRORS

BadAlloc
BadValue

SEE ALSO

XSetStandardProperties, XGetFontProperty, XRotateWindowProperties, XDeleteProperty, XChangeProperty, XGetWindowProperty, XListProperties, XGetAtomName.

NAME

XIntersectRegion — compute the intersection of two regions.

SYNOPSIS

```
XIntersectRegion(sra, srb, dr)
  Region sra, srb;
  Region dr;                    /* RETURN */
```

ARGUMENTS

sra	
srb	Specify the two regions with which to perform the computation.
dr	Returns the result of the computation.

DESCRIPTION

XIntersectRegion generates a regions that is the intersection of two regions.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XKeycodeToKeysym — convert key code to keysym.

SYNOPSIS

```
KeySym XKeycodeToKeysym(display, keycode, index)
  Display *display;
  KeyCode keycode;
  int index;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
keycode	Specifies the keycode.
index	Specifies which keysym for that keycode to return.

DESCRIPTION

XKeycodeToKeysym returns the *KeySym* defined for the specified *keycode*. *XKeycodeToKeysym* uses internal Xlib tables, which already have converted uppercase to lowercase. *index* specifies which keysym in the array of keysyms corresponding to a keycode should be returned.

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping, IsKeypadKey, IsCursorKey, IsPFKey, IsFunctionKey, IsMiscFunctionKey, IsModifierKey.

NAME

XKeysymToKeycode — convert a keysym to the appropriate keycode.

SYNOPSIS

```
KeyCode XKeysymToKeycode(display, keysym_kcode)
  Display *display;
  Keysym keysym_kcode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
keysym_kcode	Specifies the keysym that is to be searched for.

DESCRIPTION

XKeysymToKeycode returns the *KeyCode* corresponding to the specified *KeySym* symbol in the current mapping. If the specified *Keysym* is not defined for any keycode, *XKeysymToKeycode* returns zero (0).

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping, IsKeypadKey, IsCursorKey, IsPFKey, IsFunctionKey, IsMiscFunctionKey, IsModifierKey.

NAME

XKeysymToString — convert *KeySym* symbol to ASCII.

SYNOPSIS

char *XKeysymToString(*keysym_str*)
  KeySym *keysym_str*;

ARGUMENTS

keysym_str	Specifies the *KeySym* that is to be converted.

DESCRIPTION

XKeysymToString converts a *KeySym* symbol (a number) into a character string. The returned string is in a static area and must not be modified. If the specified *KeySym* is not defined, *XKeysymToString* returns *NULL*. For example, *XKeysymToString* converts **XK_SHIFT** to "XK_SHIFT".

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping, IsKeypadKey, IsCursorKey, IsPFKey, IsFunctionKey, IsMiscFunctionKey, IsModifierKey.

NAME

XKillClient — destroy a client or its remaining resources.

SYNOPSIS

```
XKillClient(display, resource)
  Display *display;
  XID resource;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
resource	Specifies any resource created by the client you want to destroy, or the constant *AllTemporary*.

DESCRIPTION

If a valid resource is specified, *XKillClient* forces a close-down of the client that created the resource. If the client has already terminated in either *RetainPermanent* or *RetainTemporary* mode, all of the client's resources are destroyed. If *AllTemporary* is specified, then the resources of all clients that have terminated in *RetainTemporary* are destroyed.

ERRORS

BadValue

SEE ALSO

XSetCloseDownMode

NAME

XListExtensions — return list of all extensions to X supported by the server.

SYNOPSIS

```
char **XListExtensions(display, nextensions)
  Display *display;
  int *nextensions;            /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
nextensions	Returns the number of extensions in the returned list.

DESCRIPTION

XListExtensions lists all the X extensions supported by the current server. The extension names will be in the ISO LATIN-1 encoding, and upper/lower case is important.

SEE ALSO

XQueryExtension, XFreeExtensionList.

NAME

XListFonts — return a list of the available font names.

SYNOPSIS

```
char **XListFonts(display, pattern, maxnames, actual_count)
  Display *display;
  char *pattern;
  int maxnames;
  int *actual_count;          /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
pattern	Specifies the string associated with the font names you want returned. You can specify any string, an asterisk (*), or a question mark. The asterisk indicates a wildcard for any number of characters and the question mark indicates a wildcard for a single character. The pattern should use the ISO Latin-1 encoding, but upper/lower case is not important.
maxnames	Specifies the maximum number of names that are to be in the returned list.
actual_count	Returns the actual number of font names in the list.

DESCRIPTION

XListFonts returns a list of font names that match the string *pattern*. Each string is terminated by *NULL*. The maximum number of names returned in the list depends on the value you passed to *maxnames*. The function returns the actual number of font names in *actual_count*. The client should call *XFreeFontNames* when done with this list to free the memory.

The font search path (the order in which font names are compared to *pattern*) is set by *XSetFontPath*.

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XListFontsWithInfo — obtain the names and information about loaded fonts.

SYNOPSIS

```
char **XListFontsWithInfo(display, pattern, maxnames, count,
info)
  Display *display;
  char *pattern;            /* null-terminated */
  int maxnames;
  int *count;               /* RETURN */
  XFontStruct **info;       /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
pattern	Specifies the string associated with the font names 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.
maxnames	Specifies the maximum number of names that are to be in the returned list.
count	Returns the actual number of matched font names.
info	Returns the font information. *XListFontsWithInfo* provides enough space for *maxnames* pointers.

DESCRIPTION

XListFontsWithInfo returns a list of font names that match the specified pattern and a list of their associated font information. The list of names is limited to size specified by the *maxnames* argument. To free the allocated name array, the client should call *XFreeFontNames.* To free the font information array, the client should call *XFreeFontInfo.*

The information returned for each font is identical to what *XQueryFont* would return, except that the per-character metrics (*lbearing, rbearing, width, ascent, descent* for single characters) are not returned.

If *XListFontsWithInfo* fails for any reason, it returns NULL.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;           /* hook for extension to hang data*/
  Font fid;                     /* Font id for this font*/
  unsigned direction;           /* hint about direction the font is painted*/
  unsigned min_char_or_byte2;/* first character*/
  unsigned max_char_or_byte2;/* last character*/
  unsigned min_byte1;           /* first row that exists*/
  unsigned max_byte1;           /* last row that exists*/
  Bool all_chars_exist;         /* flag if all characters have non-zero size*/
  unsigned default_char;        /* char to print for undefined character*/
  int n_properties;             /* how many properties there are*/
  XFontProp *properties;        /* pointer to array of additional properties*/
  XCharStruct min_bounds;       /* minimum bounds over all existing char*/
  XCharStruct max_bounds;       /* minimum bounds over all existing char*/
  XCharStruct *per_char;        /* first_char to last_char information*/
  int ascent;                   /* logical extent above baseline for spacing*/
  int descent;                  /* logical descent below baseline for spacing*/
} XFontStruct;
```

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XListHosts — obtain a list of hosts having access to this display.

SYNOPSIS

```
XHostAddress *XListHosts(display, nhosts, state)
  Display *display;
  int *nhosts;                /* RETURN */
  Bool *state;                /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
nhosts	Returns the number of hosts currently in the access control list.
state	Returns the state of access to the control list at connection setup. *True* if enabled, *False* is disabled.

DESCRIPTION

XListHosts returns the current access control list as well as whether the use of the list was enabled or disabled when this client connected to the display. *XListHosts* allows a program to find out what machines can make connections, by looking at the list of host structures. This *XHostAddress* list should be freed with *XFree* when it is no longer needed.

If *XListHosts* fails for any reason, it returns NULL.

STRUCTURES

```
typedef struct {
    int family;
    int length;
    char *address;
} XHostAddress;
```

SEE ALSO

XAddHost, XAddHosts, XRemoveHost, XRemoveHosts, XDisableAccessControl, XEnableAccessControl, XSetAccessControl.

NAME

XListInstalledColormaps — get list of installed colormaps.

SYNOPSIS

```
Colormap *XListInstalledColormaps(display, w, num)
  display *display;
  Window w;
  int *num;                    /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window for whose screen you want the list of currently installed colormaps.
num	Returns the number of currently installed colormaps in the returned list.

DESCRIPTION

XListInstalledColormaps returns a list of the currently installed colormaps for the screen of the specified window. The order in the list is not significant. There is no distinction in the list between colormaps actually being used by windows and colormaps no longer in use which have not yet been freed or destroyed. The allocated list should be freed using *XFree,* when it is no longer needed.

ERRORS

BadWindow

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XFreeColormap, XGetStandardColormap, XInstallColormap, XUninstallColormap, XSetStandardColormap, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XListProperties — get property list for window.

SYNOPSIS

```
Atom *XListProperties(display, w, num_prop)
  Display *display;
  Window w;
  int *num_prop;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window whose property list you want.
num_prop	Returns the length of the properties array.

DESCRIPTION

XListProperties returns a pointer to an array of atom properties that are defined for the specified window. To free the memory allocated by this function, use *XFree.*

When *XListProperties* fails, it returns NULL and sets the *num_prop* argument to 0.

ERRORS

BadWindow

SEE ALSO

XSetStandardProperties, XGetFontProperty, XRotateWindowProperties, XDeleteProperty, XChangeProperty, XGetWindowProperty, XGetAtomName, XInternAtom.

NAME

XLoadFont — load font if not already loaded; get font ID.

SYNOPSIS

```
Font XLoadFont(display, name)
  Display *display;
  char *name;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
name	Specifies the name of the font in a null terminated string. The font name uses ISO Latin-1 encoding, but upper/lower case is not important.

DESCRIPTION

XLoadFont loads a font into the server if it has not already been loaded by another client. *XLoadFont* returns the font ID or, if it was unsuccessful, a zero (0), and generates a *BadName* error. When the font is no longer needed, the client should call *XUnloadFont*. Fonts are not associated with a particular screen. Once the ID is available, it can be set in the *font* member of any GC, and thereby used in subsequent drawing requests.

Font information is usually necessary for locating the text. Call *XLoadFontWithInfo* to get the info at the time you load the font, or call *XQueryFont* if you used *XLoadFont* to load the font.

ERRORS

BadAlloc	
BadName	Font name specified does not identify an available font.

SEE ALSO

XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XLoadQueryFont — load a font and fill information structure.

SYNOPSIS

```
XFontStruct *XLoadQueryFont(display, name)
  Display *display;
  char *name;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
name	Specifies the name of the font. This name is a null terminated string.

DESCRIPTION

XLoadQueryFont performs a *XLoadFont* and *XQueryFont* in a single operation. *XLoadQueryFont* provides the easiest way to get character size tables for placing a proportional font. That is, *XLoadQueryFont* both opens (loads) the specified font and returns a pointer to the appropriate *XFontStruct* structure. If the font does not exist, *XLoadQueryFont* returns *NULL.*

The *XFontStruct* structure consists of the font specific information and a pointer to an array of *XCharStruct* structures for each character in the font.

STRUCTURES

```
typedef struct {
    XExtData *ext_data;      /* hook for extension to hang data*/
    Font fid;                /* Font id for this font*/
    unsigned direction;      /* hint about direction the font is painted*/
    unsigned min_char_or_byte2;/* first character*/
    unsigned max_char_or_byte2;/* last character*/
    unsigned min_byte1;      /* first row that exists*/
    unsigned max_byte1;      /* last row that exists*/
    Bool all_chars_exist;    /* flag if all characters have non-zero size*/
    unsigned default_char;   /* char to print for undefined character*/
    int n_properties;        /* how many properties there are*/
    XFontProp *properties;   /* pointer to array of additional properties*/
    XCharStruct min_bounds;  /* minimum bounds over all existing char*/
    XCharStruct max_bounds;  /* minimum bounds over all existing char*/
    XCharStruct *per_char;   /* first_char to last_char information*/
    int ascent;              /* logical extent above baseline for spacing*/
    int descent;             /* logical descent below baseline for spacing*/
} XFontStruct;

typedef struct {
    short lbearing;             /* origin to left edge of raster */
    short rbearing;             /* origin to right edge of raster */
    short width;                /* advance to next char's origin */
    short ascent;               /* baseline to top edge of raster */
    short descent;              /* baseline to bottom edge of raster */
    unsigned short attributes; /* per char flags (not predefined) */
} XCharStruct;
```

ERRORS

BadAlloc
BadName

SEE ALSO

XLoadFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XLookUpAssoc — obtain data from an association table.

SYNOPSIS

```
char *XLookUpAssoc(display, table, x_id)
  Display *display;
  XAssocTable *table;
  XID x_id;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
table	Specifies the association table.
x_id	Specifies the X resource ID.

DESCRIPTION

This function is provided for compatibility with X Version 10. To use it you must include the file *<X11/X10.h>* and link with the library *-loldX*.

Association tables provide a way of storing data and accessing by ID. This information is available to all clients. *XLookUpAssoc* retrieves the data stored in an *XAssocTable* by its *XID*. If the matching *XID* can be found in the table, the routine returns the data associated with it. If the *x_id* cannot be found in the table the routine returns *NULL*.

STRUCTURES

```
typedef struct {
    XAssoc *buckets;             /* pointer to first bucket in bucket array */
    int size;                    /* table size (number of buckets) */
} XAssocTable;

typedef struct _XAssoc {
  struct _XAssoc *next;          /* next object in this bucket */
  struct _XAssoc *prev;          /* previous object in this bucket */
  Display *display;              /* display which owns the ID */
  XID x_id;                      /* X Window System ID */
  char *data;                    /* pointer to untyped memory */
} XAssoc;
```

SEE ALSO

XCreateAssocTable, XDeleteAssoc, XDestroyAssocTable, XMakeAssoc.

NAME

XLookupColor — get database and closest hardware supported RGB values from color name.

SYNOPSIS

Status XLookupColor(*display*, *cmap*, *colorname*, *rgb_db_def*, *hardware_def*)
  Display **display*;
  Colormap *cmap*;
  char **colorname*;
  XColor **rgb_db_def*, **hardware_def*; /* RETURN */

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
colorname	Specifies the color name string (for example "red"). Upper/lowercase characters are acceptable, in ISO Latin-1 encoding.
rgb_db_def	Returns the exact RGB values for the specified color name from the */usr/lib/rgb* database.
hardware_def	Returns the closest RGB values possible on the hardware.

DESCRIPTION

XLookupColor 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 possible on that screen.

XLookupColor returns 1 if *colorname* exists in the RGB data base or 0 if it does not exist.

To determine the exact RGB values, *XLookupColor* uses a data base on the X server. On SYSTEM V/68 this data base is */usr/lib/rgb*. To read the colors provided by the data base on a SYSTEM V/68-based system, see */usr/lib/rgb.txt*. The location, name, and contents of this file are operating system specific.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                   /* DoRed, DoGreen, DoBlue */
```

```
        char pad;
    } XColor;
```

ERRORS

BadColor
BadName

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XLookupKeysym — get *KeySym* corresponding to keycode in structure.

SYNOPSIS

```
KeySym XLookupKeysym(event, index)
  XKeyEvent *event;
  int index;
```

ARGUMENTS

event	Specifies the *KeyPress* or *KeyRelease* event that is to be used.
index	Specifies which *KeySym* from the list associated with the keycode in the event to return. These correspond to the modifier keys and the symbol ShiftMapIndex.

DESCRIPTION

Given a keyboard event and the *index* into the list of *KeySyms* for that keycode, *XLookupKeysym* returns the *KeySym* from the list that corresponds to the keycode in the event.

Each keycode may have a list of associated *KeySyms*, which are portable symbols representing the meanings of the key. The *index* specifies which *KeySym* in the list is desired, indicating the combination of modifier keys that are currently pressed. Therefore, the program must interpret the *state* member of the *XKeyEvent* structure to determine the *index* before calling this function. The exact mapping of modifier keys into the list of keysyms for each keycode is server-dependent beyond the fact that the first keysym corresponds to the keycode without modifier keys, and the second corresponds to the keycode with Shift pressed.

XLookupKeysym simply calls *XKeycodeToKeysym*, using arguments taken from the specified event structure.

Note that some hardware can't support *KeyRelease* events for every key. You may wish to avoid using them in your code.

STRUCTURES

```
typedef struct {
  int type;             /* of event */
  Display *display;     /* Display the event was read from */
  Window window;        /* "event" window it is reported relative to
  Window root;          /* root window that the event occured on */
  Window subwindow;     /* child window */
  Time time;            /* milliseconds */
```

```
    int x, y;               /* pointer x, y coordinates in event window */
    int x_root, y_root;     /* coordinates relative to root */
    unsigned int state;     /* key or button mask */
    unsigned int keycode;   /* detail */
    Bool same_screen;       /* same screen flag */
  } XKeyEvent;
```

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XLookupString — map key event to ASCII string, keysym, and ComposeStatus.

SYNOPSIS

```
int XLookupString(event, buffer, num_bytes, keysym, status)
  XKeyEvent *event;
  char *buffer;              /* RETURN */
  int num_bytes;
  KeySym *keysym;            /* RETURN */
  XComposeStatus *status;    /* not implemented */
```

ARGUMENTS

event	Specifies the key event to be used.
buffer	Returns the resulting string.
num_bytes	Specifies the length of the buffer. No more than *num_bytes* of translation are returned.
keysym	If this argument is not *NULL*, it specifies the keysym ID computed from the event.
status	Specifies the *XCompose* structure that contains compose key state information and that allows the compose key processing to take place. This can be *NULL* if the caller is not interested in seeing compose key sequences. Not implemented in Release 1 or 2.

DESCRIPTION

XLookupString gets an ASCII string and a keysym that are currenly mapped to the keycode in a *KeyPress* or *KeyRelease* event, using the modifier bits in the key event to deal with shift, lock and control. The *XLookupString* return value is the length of the translated string and the string's bytes are copied into the user's buffer. The length may be greater than 1 if the event's keycode translates into a keysym which was rebound with *XRebindKeysym*.

The compose *status* is not implemented in Release 1 or 2.

STRUCTURES

```
/*
 * Compose sequence status structure, used in calling XLookupString.
 */
typedef struct _XComposeStatus {
  char *compose_ptr;/* state table pointer */
```

```
          int chars_matched;/* match state */
        } XComposeStatus;

        typedef struct {
          int type;               /* of event */
          Display *display;       /* Display the event was read from */
          Window window;          /* "event" window it is reported relative to */
          Window root;            /* root window that the event occured on */
          Window subwindow;       /* child window */
          Time time;              /* milliseconds */
          int x, y;               /* pointer x, y coordinates in event window */
          int x_root, y_root;     /* coordinates relative to root */
          unsigned int state;     /* key or button mask */
          unsigned int keycode;   /* detail */
          Bool same_screen;       /* same screen flag */
        } XKeyEvent;
```

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeySym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XSetModifierMapping, XGetModifierMapping.

NAME

XLowerWindow — lower a window in the stacking order.

SYNOPSIS

```
XLowerWindow(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the window to be lowered.

DESCRIPTION

XLowerWindow lowers a window in the stacking order of its siblings so that it does not obscure any sibling windows. If the windows are regarded as overlapping sheets of paper stacked on a desk, then lowering a window is analogous to moving the sheet to the bottom of the stack, while leaving its x and y location on the desk constant. Lowering a mapped window will generate exposure events on any windows it formerly obscured.

If the *override_redirect* attribute of the window (refer to the *GSE Programmer's Guide*) is *False* and some other client has selected *SubstructureRedirectMask* on the parent, then a *ConfigureRequest* event is generated, and no further processing is performed. Otherwise, the window is lowered to the bottom of the stack.

LeaveNotify events are sent to the lowered window if the pointer was inside it, and *EnterNotify* to the window which was immediately below the lowered window at the pointer position.

ERRORS

BadWindow

SEE ALSO

XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XMakeAssoc — create entry in an association table.

SYNOPSIS

```
XMakeAssoc(display, table, x_id, data)
  Display *display;
  XAssocTable *table;
  XID x_id;
  char *data;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
table	Specifies the assoc table.
x_id	Specifies the X resource ID.
data	Specifies the data to be associated with the X resource ID.

DESCRIPTION

XMakeAssoc inserts data into an *XAssocTable* keyed on an *XID*. Association tables allow you to easily associate data with resource ID's for later retrieval by any application.

This function is provided for compatibility with X Version 10. To use it you must include the file *<X11/X10.h>* and link with the library **-loldX**.

Data is inserted into the table only once. Redundant inserts are meaningless and cause no problems. The queue in each association bucket is sorted from the lowest *XID* to the highest *XID*.

Refer to the *GSE Programmer's Guide* for further explanation.

STRUCTURE

```
typedef struct {
  XAssoc *buckets;/* pointer to first bucket in bucket array */
  int size;/* table size (number of buckets) */
} XAssocTable;

typedef struct _XAssoc {
  struct _XAssoc *next;/* next object in this bucket */
  struct _XAssoc *prev;/* previous object in this bucket */
  Display *display;/* display which owns the ID */
```

```
          XID x_id;/* X Window System ID */
          char *data;/* pointer to untyped memory */
        } XAssoc;
```

SEE ALSO

XCreateAssocTable, XDeleteAssoc, XDestroyAssocTable, XLookUpAssoc.

NAME

XMapRaised — map a window on top of its siblings.

SYNOPSIS

```
XMapRaised(display, w)
  Display  *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XMapRaised marks a window as eligible to be displayed. It will actually be displayed if its ancestors are mapped, it is on top of sibling windows, and it is not obscured by unrelated windows. *XMapRaised* is similar to *XMapWindow*, except it additionally raises the specified window to the top of the stack among its siblings. Mapping an already mapped window with *XMapRaised* raises the window. See *XMapWindow* for further details.

ERRORS

BadWindow

SEE ALSO

XMapSubwindows, XMapWindow, XUnmapSubwindows, XUnmapWindow.

NAME

XMapSubwindows — map all subwindows.

SYNOPSIS

```
XMapSubwindows (display, w)
  Display  *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XMapSubwindows maps all subwindows of a window in top-to-bottom stacking order. *XMapSubwindows* also generates an *Expose* event on each newly displayed window. This is much more efficient than mapping many windows one at a time, as much of the work need only be performed once for all of the windows rather than for each window.

ERRORS

BadWindow

SEE ALSO

XMapRaised, XMapWindow, XUnmapSubwindows, XUnmapWindow.

NAME

XMapWindow — map a window.

SYNOPSIS

```
XMapWindow(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XMapWindow maps a window, making it eligible for display depending on its stacking order among its siblings, the mapping status of its ancestors, and the placement of other visible windows. If all the ancestors are mapped, and it is not obscured by siblings higher in the stacking order, the window and all of its mapped subwindows are displayed.

Mapping a window that has an unmapped ancestor does not display the window but marks it as eligible for display when its ancestors become mapped. Mapping an already mapped window has no effect (it does not raise the window).

If the window is opaque, *XMapWindow* generates *Expose* events on each opaque window that it causes to become displayed. If the client first maps the window, then paints the window, then begins processing input events, the window is painted twice. To avoid this, the client should use either of two strategies:

1. Map the window, call *XSelectInput* for exposure events, wait for the first *Expose* event, and repaint the window(s) explicitly.
2. Call *XSelectInput* for exposure events, map, and process input events normally. Exposure events are generated for each window that has appeared on the screen, and the client's normal response to an *Expose* event should be to repaint the window.

The latter method is preferred as it usually leads to simpler programs. If you fail to wait for the *Expose* event in the first method, it can cause incorrect behavior with certain window managers that intercept the request.

ERRORS

BadWindow

SEE ALSO

XMapRaised, XMapSubwindows, XUnmapSubwindows, XUnmapWindow.

NAME

XMaskEvent — remove next event that matches passed mask.

SYNOPSIS

```
XMaskEvent(display, event_mask, rep)
  Display *display;
  long event_mask;
  XEvent *rep;                 /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event_mask	Specifies the event mask. See *XSelectInput* for a complete list of event masks.
rep	Returns the event removed from the input queue.

DESCRIPTION

XMaskEvent removes the next event in the queue which matches the passed mask. The event is copied into an *XEvent* supplied by the caller. Other events in the queue are not discarded. If no such event has been queued, *XMaskEvent* flushes the output buffer and waits until one is received. Use *XCheckMaskEvent* if you do not wish to wait.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XMatchVisualInfo — obtain the visual information that matches the desired depth and class.

SYNOPSIS

```
Status XMatchVisualInfo(display, screen, depth, class, vinfo)
  Display *display;
  int screen;
  int depth;
  int class;
  XVisualInfo *vinfo;        /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
screen	Specifies the screen.
depth	Specifies the desired depth of the visual.
class	Specifies the desired class of the visual, such as *PseudoColor* or *TrueColor*.
vinfo	Returns the matched visual information.

DESCRIPTION

XMatchVisualInfo returns the visual information for a visual that matches the specified *depth* and *class* for a screen. Because multiple visuals that match the specified *depth* and *class* can exist, the exact visual chosen is undefined.

If a visual is found, this function returns *True* and the information on the visual is returned to *vinfo*. Otherwise, if a visual is not found, it returns *False*.

Refer to the *GSE Programmer's Guide* for a description of visuals.

STRUCTURES

```
typedef struct {
  Visual *visual;
  VisualID visualid;
  int screen;
  unsigned int depth;
  int class;
  unsigned long red_mask;
  unsigned long green_mask;
  unsigned long blue_mask;
  int colormap_size;
  int bits_per_rgb;
} XVisualInfo;
```

SEE ALSO

XGetVisualInfo, DefaultVisual.

NAME

XMoveResizeWindow — change size and location of window.

SYNOPSIS

```
XMoveResizeWindow(display, w, x, y, width, height)
  Display *display;
  Window w;
  int x, y;
  int width, height;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of the window to be reconfigured.
x *y*	Specify the x and y coordinates. These coordinates define the new position of the upper left corner of the window relative to its parent.
width *height*	Specify the width and height. These arguments define the interior size of the window.

DESCRIPTION

XMoveResizeWindow moves or resizes a window or both. Configuring a mapped window may lose its contents and generate an *Expose* event on that window depending on the *bit_gravity* and backing store attributes. Configuring a window may generate exposure events on windows that the window formerly obscured, depending on the new size and location parameters.

If the *override_redirect* attribute of the window is *False* (refer to the *GSE Programmer's Guide*) and some other client has selected *SubstructureRedirectMask* on the parent, then a *ConfigureRequest* event is generated, and no further processing is performed. Otherwise, the window size is changed. *XMoveResizeWindow* does not raise the window.

ERRORS

BadMatch
BadValue
BadWindow

SEE ALSO

XLowerWindow, *XRaiseWindow*, *XCirculateSubwindows*, *XCirculateSubwindowsDown*, *XCirculateSubwindowsUp*, *XRestackWindows*, *XMoveWindow*, *XResizeWindow*, *XReparentWindow*, *XConfigureWindow*, *XQueryTree*.

NAME

XMoveWindow — move a window.

SYNOPSIS

```
XMoveWindow(display, w, x, y)
  Display *display;
  Window w;
  int x, y;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window to be moved.
x	
y	Specify the x and y coordinates. These coordinates define the new location of the top left pixel of the window's border (or the window itself, if it has no border).

DESCRIPTION

XMoveWindow changes the position of the origin of the specified window relative to its parent. *XMoveWindow* does not change the mapping state, size, or stacking order of the window. Moving a mapped window will lose its contents if:

- Its *background_pixmap* attribute is *ParentRelative.*
- The window is obscured by non-children and no backing store exists.

If the contents are lost, exposure events will be generated for the window and any mapped subwindows. Moving a mapped window will generate exposure events on any formerly obscured windows.

If the *override_redirect* attribute of the window is *False* (refer to the *GSE Programmer's Guide*) and some other client has selected *SubstructureRedirectMask* on the parent, then a *ConfigureRequest* event is generated, and no further processing is performed.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XNewModifiermap — create a keyboard modifier mapping structure.

SYNOPSIS

```
XModifierKeymap XNewModifiermap(max_keys_per_mod)
      int max_keys_per_mod;
```

ARGUMENTS

max_keys_per_mod
Specifies the maximum number of keycodes assigned to any of the modifiers in the map.

DESCRIPTION

XNewModifiermap returns a *XModifierKeymap* structure and allocates the needed space. This function is used when more than one *XModifierKeymap* structure is needed. *max_keys_per_mod* depends on the server and should be gotten from the *XModifierKeymap* returned by *XGetModifierMapping*.

STRUCTURES

```
typedef struct {
        int max_keypermod;    /* server's max number of keys per modifier */
        KeyCode *modifiermap;/* An 8 by max_keypermod array
                              * of the modifiers */
} XModifierKeymap;
```

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XNextEvent — get next event of any type or window.

SYNOPSIS

```
XNextEvent(display, report)
  Display *display;
  XEvent *report;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
report	Returns the event removed from the input queue.

DESCRIPTION

XNextEvent removes an input event from the head of the event queue and copies it into an *XEvent* supplied by the caller. If the event queue is empty, *XNextEvent* flushes the output buffer and waits (blocks) until an event is received. Use *XCheckNextEvent* if you do not want to wait.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XNoOp — send a NoOp to exercise connection with server.

SYNOPSIS

```
XNoOp (display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XNoOp sends a *NoOperation* request to the X server, thereby exercising the connection. This request can be used to measure the response time of the network connection. *XNoOp* does not flush the output buffer.

SEE ALSO

XFree, XOpenDisplay, XCloseDisplay, DefaultScreen.

NAME

XOffsetRegion — change offset of region.

SYNOPSIS

```
XOffsetRegion(r, dx, dy)
  Region r;
  int dx, dy;
```

ARGUMENTS

r	Specifies the region.
dx *dy*	Specify the amount to change the offset of the specified region.

DESCRIPTION

XOffsetRegion changes the offset of the region the specified amounts in the x and y directions.

Regions are located using an offset from an arbitrarily chosen point (the "region origin") which is common to all regions. It is up to the application to interpret the location of the region relative to a drawable. If the region is to be used as a *clip_mask* by calling *XSetRegion,* the top left corner of the region relative to the drawable used in the graphics request will be at *(xoffset + clip_x_origin, yoffset + clip_y_origin),* where *xoffset* and *yoffset* are the offset of the region and *clip_x_origin* and *clip_y_origin* are elements of the GC used in the graphics request.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XOpenDisplay — connect a client program to an X server.

SYNOPSIS

Display *XOpenDisplay(*display_name*)
  char **display_name*;

ARGUMENTS

display_name	Specifies the display name, which determines the hardware display and communications domain to be used. See description below.

DESCRIPTION

The *XOpenDisplay* routine connects the client to the server controlling the hardware display through TCP, SYSTEM V/68, or DECnet streams.

If *display_name* is *NULL*, it defaults to the DISPLAY environment variable on SYSTEM V/68 systems. On non-SYSTEM V/68 systems, see that operating system's Xlib manual for the default *display_name*. The *display_name* or DISPLAY environment variable is a string that has the format *hostname*:*server* or *hostname*:*server*.*screen*. For example, *frog:0.2* would specify screen 2 of server *0* on the machine *frog*.

hostname	Specifies the name of the host machine on which the display is physically connected. You follow the hostname with either a single colon (:) or a double colon (::), which determines the communications domain to use. Any or all of the communication protocols can be used simultaneously on a server built to support them.

- If *hostname* is a host machine name and a single colon (:) separates the hostname and display number, *XOpenDisplay* connects the hardware display to TCP streams.
- If *hostname* is "unix" and a single colon (:) separates it from the display number, *XOpenDisplay* connects the hardware display to SYSTEM V/68 domain IPC streams.
- If *hostname* is a host machine name and a double colon (::) separates the hostname and display number, *XOpenDisplay* connects the hardware display to DECnet streams. To use DECnet, however, you must build all software for DECnet. A single X server will accept both TCP and DECnet connections if it has been built for DECnet.

server Specifies the number of the server on its host machine. This display number may be followed by a period (.).

screen Specifies the number of the default screen on *server*. Multiple screens can be connected to (controlled by) a single X server, but they are used as a single display by a single user. *screen* merely sets an internal variable that is returned by the *DefaultScreen* macro. If *screen* is omitted, it defaults to *0* structure that is defined in <*X11/Xlib.h*>.

If successful, *XOpenDisplay* returns a pointer to a *Display* This structure provides many of the specifications of the server and its screen(s). If *XOpenDisplay* does not succeed, it returns a *NULL*.

After a successful call to *XOpenDisplay*, all of the screens on the server may be used by the application. The screen number specified in the *display_name* argument serves only to specify the value that will be returned by the *DefaultScreen* macro. After opening the display, you can use the *ScreenCount* macro to determine how many screens are available. Then you can reference each screen with integer values between 0 and the value returned by *ScreenCount*.

STRUCTURES

```
/*
 * Display datatype maintaining display specific data.
 */
typedef struct _XDisplay {
  XExtData *ext_data;            /* hook for extension to hang data */
  struct _XDisplay *next;        /* next open Display on list */
  int fd;                        /* Network socket. */
  int lock;                      /* is someone in critical section */
  int proto_major_version;       /* major version of server's X protocol */
  int proto_minor_version;       /* minor version of server's X protocol */
  char *vendor;                  /* vendor of the server hardware */
        long resource_base;      /* resource ID base */
  long resource_mask;            /* resource ID mask bits */
  long resource_id;              /* allocator current ID */
  int resource_shift;            /* allocator shift to correct bits */
  XID (*resource_alloc)();       /* allocator function */
  int byte_order;                /* screen byte order, LSBFirst, MSBFirst */
  int bitmap_unit;               /* padding and data requirements */
  int bitmap_pad;                /* padding requirements on bitmaps */
  int bitmap_bit_order;          /* LeastSignificant or MostSignificant */
  int nformats;                   /* number of pixmap formats in list */
  ScreenFormat *pixmap_format; /* pixmap format list */
  int vnumber;                    /* Xlib's X protocol version number. */
  int release;                    /* release of the server */
  struct _XSQEvent *head, *tail;/* Input event queue. */
  int qlen;                       /* Length of input event queue */
  int last_request_read;          /* sequence number of last event read NI */
  int request;                    /* sequence number of last request. */
  char *last_req;                 /* beginning of last request, or dummy */
  char *buffer;                   /* Output buffer starting address. */
  char *bufptr;                   /* Output buffer index pointer. */
  char *bufmax;                   /* Output buffer maximum+1 address. */
#ifdef SHMLINK
  shmBufPtr firstShmBuf;          /* first shm buffer */
  shmBufPtr lastShmBuf;           /* last shm buffer */
  char *shmRegion;                /* Address of shm region */
  int shmId;                      /* shm id for shm region */
#endif SHMLINK
  unsigned max_request_size;   /* maximum number 32 bit words in request*/
  struct _XrmResourceDataBase *db;
  int (*synchandler)();           /* Synchronization handler */
  char *display_name;             /* "host:display" string used on this connect*
  int default_screen;             /* default screen for operations */
  int nscreens;                   /* number of screens on this server*/
  Screen *screens;                /* pointer to list of screens */
  int motion_buffer;              /* size of motion buffer */
  Window current;                 /* for use internally for Keymap notify */
  int min_keycode;                /* minimum defined keycode */
  int max_keycode;                /* maximum defined keycode */
  KeySym *keysyms;                /* This server's keysyms */
  XModifierKeymap *modifiermap;/* This server's modifier keymap */
  int keysyms_per_keycode;        /* number of rows */
  char *xdefaults;                /* contents of defaults from server */
  char *scratch_buffer;           /* place to hang scratch buffer */
  unsigned long scratch_length;/* length of scratch buffer */
  int ext_number;                 /* extension number on this display */
  _XExtension *ext_procs;         /* extensions initialized on this display */
  /*
   * the following can be fixed size, as the protocol defines how
   * much address space is available.
   * While this could be done using the extension vector, there
   * may be MANY events processed, so a search through the extension
   * list to find the right procedure for each event might be
   * expensive if many extensions are being used.
   */
  int (*event_vec[128])();      /* vector for wire to event */
  int (*wire_vec[128])();       /* vector for event to wire */
} Display;

/*
 * Information about the screen.
 */
typedef struct {
  XExtData *ext_data;           /* hook for extension to hang data */
  struct _XDisplay *display;    /* back pointer to display structure */
  Window root;                  /* Root window id. */
  int width, height;            /* width and height of screen */
  int mwidth, mheight;          /* width and height of  in millimeters */
  int ndepths;                  /* number of depths possible */
  Depth *depths;                /* list of allowable depths on the screen */
  int root_depth;               /* bits per pixel */
  Visual *root_visual;          /* root visual */
  GC default_gc;                /* GC for the root root visual */
  Colormap cmap;                /* default colormap */
  unsigned long white_pixel;
  unsigned long black_pixel;    /* White and Black pixel values */
  int max_maps, min_maps;       /* max and min colormaps */
  int backing_store;            /* Never, WhenMapped, Always */
  Bool save_unders;
  long root_input_mask;         /* initial root input mask */
} Screen;

/*
 * Format structure; describes ZFormat data the screen will understand.
 */
typedef struct {
  XExtData *ext_data;           /* hook for extension to hang data */
  int depth;                    /* depth of this image format */
  int bits_per_pixel;           /* bits/pixel at this depth */
  int scanline_pad;             /* scanline must padded to this multiple */
```

```
    } ScreenFormat;
```

SEE ALSO

XFree, XCloseDisplay, XNoOp, DefaultScreen.

NAME

XParseColor — lookup or translate RGB values from ASCII color name or hexadecimal.

SYNOPSIS

```
Status XParseColor(display, colormap, spec, rgb_db_def)
  Display *display;
  Colormap colormap;
  char *spec;
  XColor *rgb_db_def;          /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
spec	Specifies the color specification, either as a color name or as hexadecimal coded in ASCII (see below). Upper/lowercase characters are acceptable. The string must be null-terminated.
rgb_db_def	Returns the RGB values corresponding to the specified color name or hexadecimal specification, and sets its *DoRed, DoGreen* and *DoBlue* flags.

DESCRIPTION

XParseColor returns the RGB values corresponding to the English color name or hexadecimal values specified, by looking up the color name in the color database, or translating the hexadecimal code into separate RGB values. It takes a string specification of a color, typically from a command line or *XGetDefault* option, and returns the corresponding red, green, and blue values, suitable for a subsequent call to *XAllocColor* or *XStoreColor*. *spec* can be either as an English color name (as in *XAllocNamedColor*) or as an initial sharp sign character followed by a hexadecimal specification in one of the following formats:

```
#RGB                        (one character  per color)
#RRGGBB                     (two characters per color)
#RRRGGGBBB                  (three characters per color)
#RRRRGGGGBBBB               (four characters per color)
```

Where R, G, and B represent single hexadecimal digits (upper or lower case).

The hexadecimal strings must be null-terminated so that *XParseColor* knows when it has reached the end. When fewer than 16 bits each are specified, they represent the most significant bits of the value. For example, **#3a7** is the same as **#3000a0007000**. The colormap is used to determine which screen to look up the color on. The screen's default colormap is a reliable choice.

This routine will fail and return 0 status if the initial character is a sharp sign but the string otherwise fails to fit one of the above formats, or if the initial character is not a sharp sign and the named color does not exist in the server's database.

Status is 0 on failure, 1 on success.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                  /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadColor

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XQueryColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XParseGeometry — generate position and size from standard window geometry string.

SYNOPSIS

```
int XParseGeometry(parsestring, x, y, width, height)
  char *parsestring;
  int *x, *y, *width, *height;/* RETURN */
```

ARGUMENTS

parsestring	Specifies the string you want to parse.
x *y*	Return the x and y coordinates (offsets) from the string.
width *height*	Return the width and height from the string.

DESCRIPTION

By convention, X applications use a standard string to indicate window size and placement. *XParseGeometry* makes it easy to conform to this standard because it allows you to parse the standard window geometry string. Specifically, this function lets you parse strings of the form:

=<*width*>**x**<*height*>{+-}<*xoffset*>{+-}<*yoffset*>

The items in this string map into the arguments associated with this function.

XParseGeometry returns a bitmask that indicates which of the four values (*width, height, xoffset,* and *yoffset*) were actually found in the string, and whether the *x* and *y* values are negative. The bits are represented by these constants: *XValue, YValue, WidthValue, HeightValue, XNegative,* and *YNegative,* and are defined in <*X11/Xutil.h*>. For each value found, the corresponding argument is updated and the corresponding bitmask element set; for each value not found, the argument is left unchanged, and the bitmask element is not set.

SEE ALSO

XGeometry, XTranslateCoordinates.

NAME

XPeekEvent — get event without removing it from the queue.

SYNOPSIS

```
XPeekEvent(display, report)
  Display *display;
  XEvent *report;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
report	Returns the event peeked from the input queue.

DESCRIPTION

XPeekEvent peeks at an input event from the head of the event queue and copies it into an *XEvent* supplied by the caller, without removing it from the input queue. If the queue is empty, *XPeekEvent* flushes the output buffer and waits (blocks) until an event is received. If you do not want to wait, use the *QLength* macro to determine if there are any events to peek at, or use *XPeekIfEvent*. In Release 2, *XEventsQueued* can perform the function of either *QLength* or *XPending* and more.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XPeekIfEvent — get event without removing it from the queue; do not wait.

SYNOPSIS

```
XPeekIfEvent(display, event, predicate, args)
  Display *display;
  XEvent *event;            /* RETURN */
  Bool (*predicate)();
  char *args;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event	Returns the matched event.
predicate	Specifies the procedure to be called to determine if each event that arrives in the queue is the desired one.
args	Specifies the user-specified arguments that will be passed to the predicate procedure.

DESCRIPTION

XPeekIfEvent returns an event only when the specified predicate procedure returns *True* for the event. The event is copied into *event* but not removed from the queue. The specified predicate is called each time an event is added to the queue.

XPeekIfEvent flushes the output buffer if no matching events could be found on the queue, and then waits for the next matching event.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XPending — flush the output buffer and return the number of pending input events.

SYNOPSIS

```
int XPending(display)
  Display *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XPending returns the number of input events that have been received from the server, but not yet removed from the queue. If there are no events on the queue, *XPending* flushes the output buffer, and returns the number of events transferred to the input queue as a result of the flush.

The *QLength* macro returns the number of events on the queue, but without flushing the output buffer first.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XSynchronize, XSendEvent, QLength.

NAME

XPointInRegion — determine if a point is inside a region.

SYNOPSIS

```
int XPointInRegion(r, x, y)
  Region r;
  int x, y;
```

ARGUMENTS

r	Specifies the region.
x *y*	Specify the x and y coordinates of the point relative to the region origin.

DESCRIPTION

XPointInRegion returns non-zero if the point *x, y* is contained in the region *r*. The boundary is considered inside the region.

Regions are located using an offset from an arbitrarily chosen point (the "region origin") which is common to all regions. It is up to the application to interpret the location of the region relative to a drawable.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XPolygonRegion — generate a region from points.

SYNOPSIS

```
Region XPolygonRegion(points, n, fill_rule)
  XPoint points[];
  int n;
  int fill_rule;
```

ARGUMENTS

points	Specifies a pointer to an array of points.
n	Specifies the number of points in the polygon.
fill_rule	Specifies whether areas overlapping an odd number of times should be part of the region (*WindingRule*) or not part of the region (*EvenOddRule*). Refer to the *GSE Programmer's Guide* for a description of the fill rule.

DESCRIPTION

XPolygonRegion creates a region defined by connecting the specified points, and returns a pointer to be used to refer to the region.

Regions are located relative to an arbitrarily chosen point (the "region origin") which is common to all regions. In *XPolygonRegion,* the coordinates specified in *points* are relative to the region origin. By specifying all points relative to the drawable in which they will be used, the region origin can be coincident with the drawable origin. It is up to the application whether to interpret the location of the region relative to a drawable or not.

If the region is to be used as a *clip_mask* by calling *XSetRegion,* the top-left corner of region relative to the drawable used in the graphics request will be at *(xoffset + clip_x_origin, yoffset + clip_y_origin),* where *xoffset* and *yoffset* are the offset of the region (if any) and *clip_x_origin* and *clip_y_origin* are elements of the GC used in the graphics request.

- *EvenOddRule*
  Areas overlapping an odd number of times *are not* part of the region.

- *WindingRule*
  Overlapping areas are always filled.

STRUCTURES

```
typedef struct {
  short x,y;
} XPoint;

/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XPutBackEvent — push event back on the input queue.

SYNOPSIS

```
XPutBackEvent(display, event)
  Display  *display;
  XEvent  *event;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
event	Specifies a pointer to the event to be requeued.

DESCRIPTION

XPutBackEvent pushes an event back onto the head of the current display's input queue (it would become the next one returned by the next *XNextEvent* call). This can be useful if you have read an event and then decide that you'd rather deal with it later. There is no limit to how many times you can call *XPutBackEvent* in succession.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XPutImage — draw an image on window or pixmap.

SYNOPSIS

```
XPutImage(display, drawable, gc, image, src_x, src_y, dst_x, dst_y,
width, height)
  Display  *display;
  Drawable drawable;
  GC gc;
  XImage  *image;
  int src_x, src_y;
  int dst_x, dst_y;
  unsigned  int width, height;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
image	Specifies the image you want combined with the rectangle.
src_x *src_y*	Specify the offset from the top left corner of *image*.
dst_x *dst_y*	Specify the x and y coordinates. These are the coordinates of the subimage, relative to the origin of the drawable, where the image will be drawn.
width *height*	Specify the width and height of the subimage. These arguments also define the dimensions of the rectangle on the drawable.

DESCRIPTION

XPutImage draws a section of an image on a rectangle in a window or pixmap. The section of the image is defined by *src_x, src_y, width,* and *height.*

XPutImage uses these graphics context components: *function, plane_mask, subwindow_mode, clip_x_origin, clip_y_origin,* and *clip_mask.* This function also uses these graphics context mode-dependent components: *foreground* and *background.*

If an *XYBitmap* format image is used, then the depth of *drawable* must be one and the image must be *XYFormat,* otherwise a *BadMatch* error is generated. The *foreground* pixel in *gc* defines the source for set bits in the image, and the *background* pixel defines the source for the zero bits.

For *XYPixmap* and *ZPixmap* format images, the depth of the image must match the depth of *drawable.* For *XYPixmap,* the image must be sent in *XYFormat.* For *ZPixmap,* the image must be sent in the *ZFormat* defined for the given depth.

STRUCTURES

```
typedef struct _XImage {
  int width, height;           /* size of image */
  int xoffset;                 /* number of pixels offset in X directi
  int format;                  /* XYBitmap, XYPixmap, ZPixmap */
  char *data;                  /* pointer to image data */
  int byte_order;              /* data byte order, LSBFirst, MSBFirst
  int bitmap_unit;             /* quant. of scanline 8, 16, 32 */
  int bitmap_bit_order;/* LSBFirst, MSBFirst */
  int bitmap_pad;              /* 8, 16, 32 either XY or ZPixmap */
  int depth;                   /* depth of image */
  int bytes_per_line;          /* accelarator to next line */
  int bits_per_pixel;          /* bits per pixel (ZPixmap) */
  char *obdata;                /* hook for the object routines to hang
  struct funcs {               /* image manipulation routines */
  struct _XImage *(*create_image)();
  int (*destroy_image)();
  unsigned long (*get_pixel)();
  int (*put_pixel)();
  struct _XImage *(*sub_image)();
  int (*add_pixel)();
  } f;
} XImage;
```

ERRORS

BadDrawable	
BadGC	
BadMatch	See Description above.
BadValue	

SEE ALSO

XDestroyImage, XGetImage, XCreateImage, XSubImage, XGetSubImage, XAddPixel, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XPutPixel — set a pixel value in an image.

SYNOPSIS

```
int XPutPixel(ximage, x, y, pixel)
  XImage *ximage;
  int x;
  int y;
  unsigned long pixel;
```

ARGUMENTS

ximage	Specifies a pointer to the image.
x	
y	Specify the x and y coordinates.
pixel	Specifies the new pixel value.

DESCRIPTION

XPutPixel overwrites the pixel in the named image with the specified pixel value. The *x* and *y* coordinates are relative to the origin (upper left [0,0]) of the image. The input pixel value must be in normalized format (that is, the Least Significant Byte (LSB) of the long is the LSB of the pixel).

STRUCTURES

```
typedef struct _XImage {
  int width, height;            /* size of image */
  int xoffset;                  /* number of pixels offset in X direction */
  int format;                   /* XYBitmap, XYPixmap, ZPixmap */
  char *data;                   /* pointer to image data */
  int byte_order;               /* data byte order, LSBFirst, MSBFirst */
  int bitmap_unit;              /* quant. of scanline 8, 16, 32 */
  int bitmap_bit_order;/* LSBFirst, MSBFirst */
  int bitmap_pad;               /* 8, 16, 32 either XY or ZPixmap */
  int depth;                    /* depth of image */
  int bytes_per_line;           /* accelarator to next line */
  int bits_per_pixel;           /* bits per pixel (ZPixmap) */
  unsigned long red_mask;/* bits in z arrangment */
  unsigned long green_mask;
  unsigned long blue_mask;
  char *obdata;                 /* hook for the object routines to hang on */
  struct funcs {                /* image manipulation routines */
  struct _XImage *(*create_image)();
  int (*destroy_image)();
```

```
          unsigned long (*get_pixel)();
          int (*put_pixel)();
          struct _XImage *(*sub_image)();
          int (*add_pixel)();
          } f;
     } XImage;
```

SEE ALSO

XDestroyImage, XPutImage, XGetImage, XCreateImage, XSubImage, XGetSubImage, XAddPixel, XGetPixel, ImageByteOrder.

NAME

XQueryBestCursor — get closest supported cursor sizes.

SYNOPSIS

```
XQueryBestCursor(display, d, width, height, rwidth, rheight)
  Display *display;
  Drawable d;
  unsigned int width, height;
  unsigned int *rwidth, *rheight;/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
d	Specifies the drawable.
width *height*	Specify the preferred width and height.
rwidth *rheight*	Return pointers to the closest supported cursor dimensions on the display hardware.

DESCRIPTION

XQueryBestCursor returns the closest cursor dimensions actually supported by the display hardware to the dimensions you specify.

Call this function if you wish to use a cursor size other than 16 x 16. *XQueryBestCursor* provides a way to find out what size cursors are actually possible on the display. It returns dimensions acceptable for *XCreatePixmapCursor*. Applications should be prepared to use smaller cursors on displays which cannot support large ones.

ERRORS

BadDrawable

SEE ALSO

XDefineCursor, XUndefineCursor, XCreateFontCursor, XCreateGlyphCursor, XCreatePixmapCursor, XFreeCursor, XRecolorCursor, XQueryBestSize.

NAME

XQueryBestSize — obtain the "best" supported cursor, tile, or stipple size.

SYNOPSIS

```
XQueryBestSize(display, class, which_screen, width, height, rwidth,
rheight)
  Display  *display;
  int class;
  Drawable which_screen;
  unsigned  int width, height;
  unsigned  int  *rwidth,  *rheight;/*  RETURN  */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
class	Specifies the class that you are interested in. Pass one of these constants: *TileShape, CursorShape,* or *StippleShape.*
which_screen	Specifies a drawable ID which tells the server which screen you want the best size for.
width *height*	Specify the width and height desired.
rwidth *rheight*	Return the closest supported width and height available for the object on the display hardware.

DESCRIPTION

XQueryBestSize returns the "fastest" or "closest" size to the specified size. For *class* of *CursorShape,* this is the closest size that can be fully displayed on the screen. For *TileShape* and *StippleShape,* this is the closest size that can be tiled or stippled "fastest."

For *CursorShape,* the drawable indicates the desired screen. For *TileShape* and *StippleShape,* the drawable indicates the screen and possibly the visual class and depth (server dependent). An *InputOnly* window cannot be used as the drawable for *TileShape* or *StippleShape* (else a *BadMatch* error occurs).

ERRORS

BadDrawable	
BadMatch	*InputOnly* drawable for *class TileShape* or *StippleShape*.
BadValue	

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XQueryBestStipple — obtain the best supported stipple shape.

SYNOPSIS

```
XQueryBestStipple(display, which_screen, width, height, rwidth,
rheight)
  Display *display;
  Drawable which_screen;
  unsigned int width, height;
  unsigned int *rwidth, *rheight;/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
which_screen	Specifies a drawable which tells the server which screen you want the best size for.
width *height*	Specify the width and height desired.
rwidth *rheight*	Return the width and height of the stipple best supported by the display hardware.

DESCRIPTION

XQueryBestStipple returns the closest stipple size that can be stippled fastest. The drawable indicates the screen and possibly the visual class and depth. An *InputOnly* window cannot be used as the drawable (else a *BadMatch* error occurs).

ERRORS

BadDrawable	
BadMatch	*InputOnly* window.

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XQueryBestTile — obtain the best supported fill tile shape.

SYNOPSIS

```
XQueryBestTile(display, which_screen, width, height, rwidth,
rheight)
  Display *display;
  Drawable which_screen;
  unsigned int width, height;
  unsigned int *rwidth, *rheight;/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
which_screen	Specifies a drawable which tells the server which screen you want the best size for.
width *height*	Specify the width and height desired.
rwidth *rheight*	Return the width and height of the tile best supported by the display hardware.

DESCRIPTION

XQueryBestTile returns the "closest" size that can be tiled "fastest." The drawable indicates the screen and possibly the visual class and depth. An *InputOnly* window cannot be used as the drawable.

ERRORS

BadDrawable	
BadMatch	*InputOnly* drawable specified.

SEE ALSO

XSetTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XQueryColor — obtains the RGB values for the specified pixel value.

SYNOPSIS

```
XQueryColor(display, cmap, colorcell_def)
  Display *display;
  Colormap cmap;
  XColor *colorcell_def;        /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap ID.
colorcell_def	Specifies the pixel value and returns the RGB contents of that colorcell.

DESCRIPTION

XQueryColor returns the RGB values stored in *cmap* for the pixel value you specified in the same *XColor* structure, and sets the *flags* member of that structure to *(DoRed* | *DoGreen* | *DoBlue)*. The values returned for an unallocated entry are undefined.

XQueryColor returns zero if it encountered some problem, or non-zero if it succeeded.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                    /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadValue	Pixel not valid index into *cmap*.
BadColor	

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColors, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XQueryColors — obtain RGB values for an array of pixel values.

SYNOPSIS

```
XQueryColors(display, cmap, colorcell_defs, ncolors)
  Display *display;
  Colormap cmap;
  XColor colorcell_defs[ncolors];
  int ncolors;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
colorcell_defs	Specifies an array of *XColor* structures.
ncolors	Specifies the number of *XColor* structures in the color definition array.

DESCRIPTION

XQueryColors returns the RGB values stored in *cmap* for each pixel value passed in the *pixel* member of each *XColor* structure, and sets the *flags* member in each *XColor* structure(s) to *(DoRed | DoGreen | DoBlue).*

XQueryColors returns zero if it encounters some problem, or non-zero if it succeeds.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                    /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadColor

BadValue	Pixel not valid index into *cmap*.

Note: if more than one pixel is in error, the one reported is arbitrary.

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XStoreColor, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XQueryExtension — get extension information.

SYNOPSIS

```
Bool XQueryExtension(display, name, major_opcode, first_event,
first_error)
  Display *display;
  char *name;
  int *major_opcode;          /* RETURN */
  int *first_event;           /* RETURN */
  int *first_error;           /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
name	Specifies the name of the desired extension. *name* should be in ISO LATIN-1 encoding, and upper/lower case is important.
major_opcode	Returns the major opcode of the extension, for use in error handling routines.
first_event	Returns the code of the first custom event type created by the extension.
first_error	Returns the code of the first custom error defined by the extension.

DESCRIPTION

XQueryExtension determines if the named extension is present, and returns *True* if it is. If so, the routines in the extension can be used just as if they were core Xlib requests, except that they may return new types of events or new error codes. The available extensions can be listed with *XListExtensions*.

The *major_opcode* for the extension is returned, if it has one. Otherwise, zero is returned. This opcode will appear in errors generated in the extension.

If the extension involves additional event types, the base event type code is returned in *first_event*. Otherwise, zero is returned in *first_event*. The format of the events is specific to the extension.

If the extension involves additional error codes, the base error code is returned in *first_error*. Otherwise, zero is returned. The format of additional data in the errors is specific to the extension.

Refer to the *GSE Programmer's Guide* for more information on using and writing extensions.

SEE ALSO

XListExtensions, XFreeExtensionList.

NAME

XQueryFont — return information about loaded font.

SYNOPSIS

```
XFontStruct  *XQueryFont(display, font_ID)
  Display  *display;
  XID font_ID;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
font_ID	Specifies either the font ID or the graphics context ID. You can declare the data type for this argument as either *Font* or *GContext* (both X IDs).

DESCRIPTION

XQueryFont returns a pointer to the *XFontStruct* structure information associated with the font. This call is needed if you loaded the font with *XLoadFont,* but need the font information to place text. *XLoadQueryFont* both loads and gets information about a font.

If *font_ID* is declared as data type *GContext* (also a resource ID), this function queries the font stored in the GC specified by this ID. However, in this case the *GContext* ID will be the ID stored in the *XFontStruct,* and you cannot use that ID in *XSetFont* or *XUnloadFont*.

Use *XFreeFontInfo* to free this data.

XQueryFont returns NULL if the specified font is not loaded or if the routine fails for some other reason.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;       /* hook for extension to hang data */
  Font fid;                 /* Font id for this font */
  unsigned direction;       /* hint about direction font is painted */
  unsigned min_char_or_byte2;  /* first character */
  unsigned max_char_or_byte2;  /* last character */
  unsigned min_byte1;       /* first row that exists */
  unsigned max_byte1;       /* last row that exists */
  Bool all_chars_exist;     /* flag if all characters have non-zero size*/
  unsigned default_char;    /* char to print for undefined character */
  int n_properties;         /* how many properties there are */
  XFontProp *properties;    /* pointer to array of additional properties*/
```

```
    XCharStruct min_bounds;  /* minimum bounds over all existing char*/
    XCharStruct max_bounds;  /* minimum bounds over all existing char*/
    XCharStruct *per_char;   /* first_char to last_char information */
    int ascent;    /* logical extent above baseline for spacing */
    int descent;   /* logical descent below baseline for spacing */
  } XFontStruct;
```

ERRORS

BadAlloc
BadFont

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XSetFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XQueryKeymap — obtain bit vector for current state of keyboard.

SYNOPSIS

```
XQueryKeymap(display, keys)
  Display *display;
  char keys[32];              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
keys	Returns an array of bytes that identifies which keys are pressed down. Each bit represents one key of the keyboard.

DESCRIPTION

XQueryKeymap returns a bit vector for the logical state of the keyboard, where each bit set to one indicates that the corresponding key is currently pressed down. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7 with the least significant bit in the byte representing key 8N. Note that the logical state may log the physical state if device event processing is frozen due to a grab.

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XStringToKeysym, XLookupKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XQueryPointer — get current pointer location.

SYNOPSIS

```
Bool XQueryPointer(display, w, root, child, root_x, root_y,
win_x, win_y, keys_buttons)
  Display *display;
  Window w;
  Window *root, *child;      /* RETURN */
  int *root_x, *root_y;     /* RETURN */
  int *win_x, *win_y;         /* RETURN */
  unsigned int *keys_buttons;       /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies a window which indicates which screen the pointer position is returned for, and *child* will be a child of this window if pointer is inside a child.
root	Returns the root window ID the pointer is currently on.
child	Returns the child window ID the pointer is located in, if any.
root_x *root_y*	Return the x and y coordinates relative to the root's origin.
win_x *win_y*	Return the x and y coordinates relative to window *w*.
keys_buttons	Returns the current state of the modifier keys and pointer buttons. This is a mask composed of the OR of any number of the following symbols: *ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask, Button1Mask, Button2Mask, Button3Mask, Button4Mask, Button5Mask.*

DESCRIPTION

XQueryPointer gets the pointer coordinates relative to a window and relative to the root window, the *root* window ID and the *child* window ID (if any) the pointer is currently in, and the current state of modifier keys and buttons.

If *XQueryPointer* returns *False,* then the pointer is not on the same screen as *w, child* is *None,* and *win_x* and *win_y* are zero. However, *root, root_x,* and *root_y* are still valid. If *XQueryPointer* returns *True,* then the pointer is on the same screen as the window *w,* and all return values are valid.

The logical state of the pointer buttons and modifier keys can lag behind their physical state if device event processing is frozen due to a grab.

ERRORS

BadWindow

SEE ALSO

XWarpPointer, XGrabPointer, XChangeActivePointerGrab, XUngrabPointer, XGetPointerMapping, XSetPointerMapping, XGetPointerControl, XChangePointerControl.

NAME

XQueryTextExtents — query server for string and font metrics.

SYNOPSIS

```
int XQueryTextExtents(display, font_ID, string, nchars, direction, ascent, descent, overall)
  Display *display;
  XID font_ID;
  char *string;
  int nchars;
  int *direction;               /* RETURN */
  int *ascent, *descent;        /* RETURN */
  XCharStruct *overall;         /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
font_ID	Specifies the appropriate font ID previously returned by *XLoadFont*, or the *GContext* that specifies the font.
string	Specifies the character string.
nchars	Specifies the number of characters in the character string.
direction	Returns the direction the string would be drawn using the specified font. Either *FontLeftToRight* or *FontRightToLeft*.
ascent	Returns the maximum ascent for the specified font.
descent	Returns the maximum descent for the specified font.
overall	Returns the overall characteristics of the string. These are the sum of the *width* measurements for each character, the maximum *ascent* and *descent*, the minimum *lbearing* added to the width of all characters up to the character with the smallest lbearing, and the maximum *rbearing* added to the width of all characters up to the character with the largest rbearing.

DESCRIPTION

XQueryTextExtents returns the dimensions in pixels that specify the bounding box of the specified string of characters in the named font, and the maximum ascent and descent for the entire font. This function queries the server and, therefore, suffers the round trip overhead that is avoided by *XTextExtents*, but it does require a filled *XFontInfo* structure.

The returned *ascent* and *descent* should usually be used to calculate the line spacing, while the *width, rbearing,* and *lbearing* members of *overall* should be used for horizontal measures. The total height of the bounding rectangle, good for any string in this font, is *ascent* + *descent.*

overall.ascent is the maximum of the ascent metrics of all characters in the string. The *overall.descent* is the maximum of the descent metrics. The *overall.width* is the sum of the character-width metrics of all characters in the string. The *overall.lbearing* is the lbearing of the character in the string with the smallest lbearing plus the width of all the characters up to but not including that character. The *overall.rbearing* is the rbearing of the character in the string with the largest lbearing plus the width of all the characters up to but not including that character.

XQueryTextExtents returns 1 on success and 0 on failure.

STRUCTURES

```
typedef struct {
  short lbearing;              /* origin to left edge of raster */
  short rbearing;              /* origin to right edge of raster */
  short width;                 /* advance to next char's origin */
  short ascent;                /* baseline to top edge of raster */
  short descent;               /* baseline to bottom edge of raster */
  unsigned short attributes;   /* per char flags (not predefined) */
} XCharStruct;
```

ERRORS

BadFont
BadGC

SEE ALSO

XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XQueryTextExtents16 — query server for string and font metrics of 16-bit character string.

SYNOPSIS

```
int XQueryTextExtents16(display, font_ID, string, nchars,
direction, ascent, descent, overall)
  Display *display;
  XID font_ID;
  XChar2b *string;
  int nchars;
  int *direction;              /* RETURN */
  int *ascent, *descent;       /* RETURN */
  XCharStruct *overall;        /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
font_ID	Specifies the appropriate font ID previously returned by *XLoadFont*, or the *GContext* that specifies the font.
string	Specifies the character string. Either *FontLefttoRight* or *FontRighttoLeft*.
nchars	Specifies the number of characters in the character string.
direction	Returns the direction of painting in the specified font.
ascent	Returns the maximum ascent for the specified font.
descent	Returns the maximum descent for the specified font.
overall	Returns the overall characteristics of the string. These are the sum of the *width* measurements for each character, the maximum *ascent* and *descent*, the minimum *lbearing* added to the width of all characters up to the character with the smallest lbearing, and the maximum *rbearing* added to the width of all characters up to the character with the largest rbearing.

DESCRIPTION

XQueryTextExtents16 returns the dimensions in pixels that specify the bounding box of the specified string of characters in the named font, and the maximum ascent and descent for the entire font. This function queries the server and, therefore, suffers the round trip overhead that is avoided by *XTextExtents16,* but it does require a filled *XFontInfo* structure.

The returned *ascent* and *descent* should usually be used to calculate the line spacing, while the *width, rbearing,* and *lbearing* members of *overall* should be used for horizontal measures. The total height of the bounding rectangle, good for any string in this font, is *ascent* + *descent.*

overall.ascent is the maximum of the ascent metrics of all characters in the string. The *overall.descent* is the maximum of the descent metrics. The *overall.width* is the sum of the character-width metrics of all characters in the string. The *overall.lbearing* is the lbearing of the character in the string with the smallest lbearing plus the width of all the characters up to but not including that character. The *overall.rbearing* is the rbearing of the character in the string with the largest lbearing plus the width of all the characters up to but not including that character.

For fonts defined with linear indexing rather than two-byte matrix indexing, the server interprets each *XChar*2b as a 16-bit number that has been transmitted with the most significant byte first. That is, byte1 of the *XChar*2b is taken as the most significant byte.

If the font has no defined default character, then undefined characters in the string are taken to have all zero metrics.

XQueryTextExtents16 returns 1 on success and 0 on failure.

STRUCTURES

```
typedef struct {                /* normal 16 bit characters are two bytes */
  unsigned char byte1;
  unsigned char byte2;
} XChar2b;

typedef struct {
  short lbearing;               /* origin to left edge of raster */
  short rbearing;               /* origin to right edge of raster */
  short width;                  /* advance to next char's origin */
  short ascent;                 /* baseline to top edge of raster */
  short descent;                /* baseline to bottom edge of raster */
  unsigned short attributes;    /* per char flags (not predefined) */
} XCharStruct;
```

ERRORS

BadFont
BadGC

SEE ALSO

XQueryTextExtents, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XQueryTree — obtains a list of children, parent, and root.

SYNOPSIS

```
Status XQueryTree(display, w, root, parent, children, nchildren)
  Display *display;
  Window w;
  Window *root;           /* RETURN */
  Window *parent;         /* RETURN */
  Window **children;      /* RETURN */
  unsigned int *nchildren; /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. For this window, *XQueryTree* will list its children, its root, its parent, and the number of children.
root	Returns the root ID for the specified window.
parent	Returns the parent window of the specified window.
children	Returns the list of children associated with the specified window.
nchildren	Returns the number of children associated with the specified window.

DESCRIPTION

XQueryTree uses its last four arguments to return the root ID, the parent ID, a pointer to a list of children and the number of children in that list, all for the specified window *w*. The *children* are listed in current stacking order, from bottom-most (first) to top-most (last). *XQueryTree* returns 0 if it fails, 1 if it succeeds.

You should deallocate the list of children with *XFree* when it is no longer needed.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow.

NAME

XRaiseWindow — raise a window to top of stacking order.

SYNOPSIS

```
XRaiseWindow(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. *XRaiseWindow* raises this window to the top of the stack.

DESCRIPTION

XRaiseWindow moves a window to the top of the stacking order among its siblings. If the windows are regarded as overlapping sheets of paper stacked on a desk, then raising a window is analogous to moving the sheet to the top of the stack, while leaving its x and y location on the desk constant.

Raising a mapped window may generate exposure events for that window and any mapped subwindows of that window that were formerly obscured.

If the *override_redirect* attribute of the window (refer to the *GSE Programmer's Guide*) is *False* and some other client has selected *SubstructureRedirectMask* on the parent, then a *ConfigureRequest* event is generated, and no further processing is performed.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XReadBitmapFile — read a bitmap from disk.

SYNOPSIS

```
int XReadBitmapFile(display, d, filename, width, height, bitmap,
x_hot, y_hot)
  Display *display;
  Drawable d;
  char *filename;
  unsigned int *width, *height;/* RETURN */
  Pixmap *bitmap;              /* RETURN */
  unsigned int *x_hot, *y_hot;/* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
d	Specifies the drawable.
filename	Specifies the file name to use. The format of the file name is operating system specific.
width *height*	Return the dimensions of the bitmap that is read.
bitmap	Returns the pixmap resource ID that is created.
x_hot *y_hot*	Return the hot spot coordinates in the file (or -1,-1 if none present).

DESCRIPTION

XReadBitmapFile reads in a file containing a pixmap of depth one (a bitmap). The file can be either in the standard X version 10 format or in the newer X version 11 bitmap format (which is only slightly different).

XReadBitmapFile creates a pixmap of the appropriate size, reads the bitmap data from the file into the pixmap. The caller must free the bitmap using *XFreePixmap* when done.

If the file cannot be opened, *XReadBitmapFile* returns *BitmapOpenFailed.* If the file can be opened but does not contain valid bitmap data, *XReadBitmapFile* returns *BitmapFileInvalid.* If insufficient working storage is allocated, *XReadBitmapFile* returns *BitmapNoMemory.* If the file is readable and valid, *XReadBitmapFile* returns *BitmapSuccess.*

Here is a Version 11 example bitmap file:

```
#define name_width 16
#define name_height 16
#define name_x_hot 8
#define name_y_hot 8
static char name_bits[] =
  {
  0xf81f, 0xe3c7, 0xcff3, 0x9ff9,     /* each data entry 16 bits */
  0xbffd, 0x33cc, 0x7ffe, 0x7ffe,
  0x7e7e, 0x7ffe, 0x37ec, 0xbbdd,
  0x9c39, 0xcff3, 0xe3c7, 0xf81f
  };
```

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XCreateBitmapFromData.

NAME

XRebindKeysym — rebind *KeySym* to string for client.

SYNOPSIS

XRebindKeysym(*display*, *keysym*, *mod_list*, *mod_count*, *string*, *num_bytes*)
  Display **display*;
  KeySym *keysym*;
  KeySym **mod_list*;
  int *mod_count*;
  unsigned char **string*;
  int *num_bytes*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
keysym	Specifies the *KeySym* to be rebound.
mod_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.
string	Specifies a pointer to the string that is to be returned by *XLookupString*.
num_bytes	Specifies the length of the string.

DESCRIPTION

XRebindKeysym binds the ASCII *string* to the specified *keysym*, so that *string* and *keysym* are returned when that key is pressed and the modifiers specified in *mod_list* are also being held down. This function rebinds the meaning of a keysym for a client. It does not redefine the keycode in the server but merely provides an easy way for long strings to be attached to keys. Note that you are allowed to rebind a KeySym that may not exist.

Refer to the *GSE Programmer's Guide* for a description of keysyms and keyboard mapping.

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XRecolorCursor — change color of cursor.

SYNOPSIS

```
XRecolorCursor(display, cursor, foreground_color, background_color)
  Display *display;
  Cursor cursor;
  XColor *foreground_color, *background_color;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cursor	Specifies the cursor ID.
foreground_color	Specifies the red, green, and blue (RGB) values for the foreground.
background_color	Specifies the red, green, and blue (RGB) values for the background.

DESCRIPTION

XRecolorCursor applies a foreground and background color to a bitmap cursor. Cursors are normally created using a single plane pixmap, composed of 0s and 1s. *XRecolorCursor* applies a pixel value to each of these bit states. If the cursor is being displayed on a screen, the change is visible immediately. On some servers, these color selections are read/write cells from the colormap, and cannot be shared by applications.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                  /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadCursor

SEE ALSO

XDefineCursor, XUndefineCursor, XCreateFontCursor, XCreateGlyphCursor, XCreatePixmapCursor, XFreeCursor, XQueryBestCursor, XQueryBestSize.

NAME

XRectInRegion — determine if rectangle resides in region.

SYNOPSIS

```
int XRectInRegion(r, x, y, width, height)
  Region r;
  unsigned int x, y, width, height;
```

ARGUMENTS

r	Specifies the region.
x *y*	Specify the x and y coordinates of the top-left corner of the rectangle relative to the region origin.
width *height*	Specify the width and height of the rectangle.

DESCRIPTION

XRectInRegion returns *RectangleIn* if the rectangle is completely contained in the region *r*, *RectangleOut* if it is completely outside, and *RectanglePart* is it is partially inside.

Regions are located using an offset from an arbitrarily chosen point (the "region origin") which is common to all regions. It is up to the application to interpret the location of the region relative to a drawable. If the region is to be used as a *clip_mask* by calling *XSetRegion,* the top-left corner of region relative to the drawable used in the graphics request will be at *(xoffset + clip_x_origin, yoffset + clip_y_origin),* where *xoffset* and *yoffset* are the offset of the region and *clip_x_origin* and *clip_y_origin* are the clip origin in the GC used.

For this function, the *x* and *y* arguments are interpreted relative to the region origin, not the drawable origin.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XRefreshKeyboardMapping — update the stored modifier and keymap information.

SYNOPSIS

```
XRefreshKeyboardMapping(event)
  XMappingEvent  *event;
```

ARGUMENTS

event	Specifies the mapping event that is to be used.

DESCRIPTION

XRefreshKeyboardMapping causes the library to update the mapping between keycodes and keysyms. This updates the client application's knowledge of the keyboard.

You usually want to call *XRefreshKeyboardMapping* when a *MappingNotify* event occurs. *MappingNotify* events occur when some client has called *XChangeKeyboardMapping*.

STRUCTURES

```
typedef struct {
  int type;
  Display *display;    /* display the event was read from */
  Window window;       /* unused */
  int request;         /* one of MappingModifier, MappingKeyboard,
                          MappingPointer */
  int first_keycode;   /* first keycode */
  int count;           /* defines range of change w. first_keycode*/
} XMappingEvent;
```

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XRemoveFromSaveSet — removes a window's children from the client's save-set.

SYNOPSIS

```
XRemoveFromSaveSet(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window whose children you want to remove from this client's save-set. This window must have been created by a client other than the client making this call.

DESCRIPTION

XRemoveFromSaveSet removes a window's children from the save-set of the calling application. Usually, this call is invoked by a window manager, using *RootWindow* macro for *w*, to remove all top-level windows on a screen from the save-set.

The save-set is a safety net for windows that have been reparented by the window manager, usually to provide a shadow or other background for each window. When the window manager dies unexpectedly, the windows in the save-set are reparented to their closest living ancestor, so that they remain alive.

This call is not necessary when a window is destroyed since destroyed windows are automatically removed from the save-set. Refer to the *GSE Programmer's Guide* for more information about save-sets.

ERRORS

BadMatch	*w* not created by some other client.
BadWindow	

SEE ALSO

XAddToSaveSet, XChangeSaveSet.

NAME

XRemoveHost — remove host from access control list.

SYNOPSIS

```
XRemoveHost(display, host)
  Display *display;
  XHostAddress *host;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
host	Specifies the network address of the machine to be removed.

DESCRIPTION

XRemoveHost removes the specified host from the access control list on the host running the server controlling the current display. The display hardware must be on the same host as the calling process in order to change the access control list.

If you remove your own machine from the access control list, you can no longer connect to that server, and there is no way back from this call other than to log out and reset the server.

The *address* data must be a valid address for the type of network in which the server operates, as specified in the *family* member.

STRUCTURES

```
typedef struct {
  int family;                /* for example AF_DNET */
  int length;                /* length of address, in bytes */
  char *address;             /* pointer to where to find the bytes
} XHostAddress;

/* constants used for family member of XHostAddress */
#define FamilyInternet         0
#define FamilyDECnet           1
#define FamilyChaos            2
```

ERRORS

BadAlloc
BadValue

SEE ALSO

XAddHost, XAddHosts, XListHosts, XRemoveHosts, XDisableAccessControl, XEnableAccessControl, XSetAccessControl.

NAME

XRemoveHosts — remove multiple hosts from the access control list.

SYNOPSIS

XRemoveHosts (*display*, *hosts*, *num_hosts*)
  Display **display*;
  XHostAddress **hosts*;
  int *num_hosts*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
hosts	Specifies the list of hosts that are to be removed.
num_hosts	Specifies the number of hosts that are to be removed.

DESCRIPTION

XRemoveHosts removes each specified host from the access control list on the local machine running the server. The display hardware must be on the same host as the client process, in order to change the access control list.

If you remove your machine from the access control list, you can no longer connect to that server, and there is no way back from this call except to log out and reset the server.

The *address* data must be a valid address for the type of network in which the server operates, as specified in the *family* member.

STRUCTURES

```
typedef struct {
  int family;                 /* for example AF_DNET */
  int length;                 /* length of address, in bytes */
  char *address;              /* pointer to where to find the bytes
} XHostAddress;

/* constants used for family member of XHostAddress */
#define FamilyInternet        0
#define FamilyDECnet          1
#define FamilyChaos           2
```

ERRORS

BadAlloc
BadValue

SEE ALSO

XAddHost, XAddHosts, XListHosts, XRemoveHost, XDisableAccessControl, XEnableAccessControl, XSetAccessControl.

NAME

XReparentWindow — change a window's parent.

SYNOPSIS

```
XReparentWindow(display, w, parent, x, y)
  Display  *display;
  Window w;
  Window parent;
  int x, y;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
parent	Specifies the parent window ID.
x	
y	Specify the coordinates of the window relative to the new parent.

DESCRIPTION

XReparentWindow modifies the window hierarchy by inserting the window *w* as a child of *parent*. This function is usually used by the window manager to put a border behind application windows.

If *w* is mapped, an *XUnmapWindow* request is performed first automatically. *w* is then removed from its current position in the hierarchy, and is inserted as a child of the specified parent. *w* is placed on top in the stacking order with respect to siblings. A *ReparentNotify* event is then generated. The *override_redirect* member of the structure returned by this event is set to either *True* or *False*. Window manager clients normally should ignore this event if this member is set to *True*.

Finally, if the window was originally mapped, an *XMapWindow* request is performed automatically.

Normal exposure processing on formerly obscured windows is performed. The server might not generate exposure events for regions from the initial unmap that are immediately obscured by the final map. The request fails if the new parent is not on the same screen as the old parent, or if the new parent is the window itself or an inferior of the window.

ERRORS

BadMatch	*parent* not on same screen as old parent of *w*.
	w has a *ParentRelative* background and *parent* is not the same depth as *w*.
	parent is *w* or an inferior of *w*.
BadWindow	

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XConfigureWindow, XQueryTree.

NAME

XResetScreenSaver — reset the screen saver.

SYNOPSIS

XResetScreenSaver (*display*)
  Display **display*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XResetScreenSaver redisplays the screen if the screen saver was activated. This may result in exposure events to all visible windows if the server cannot save the screen contents. If the screen is already active, nothing happens.

SEE ALSO

XForceScreenSaver, XActivateScreenSaver, XGetScreenSaver, XSetScreenSaver.

NAME

XResizeWindow — change a window's size.

SYNOPSIS

```
XResizeWindow(display, w, width, height)
  Display *display;
  Window w;
  int width, height;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
width *height*	Specify the new dimensions of the window.

DESCRIPTION

XResizeWindow changes the inside dimensions of the window. The border is resized to match but its width is not changed. *XResizeWindow* does not raise the window, or change its origin. Changing the size of a mapped window may lose its contents and generate an *Expose* event, depending on the *bit_gravity* attribute (refer to the *GSE Programmer's Guide*). If a mapped window is made smaller, exposure events will be generated on windows that it formerly obscured.

If the *override_redirect* attribute of the window is *False* and some other client has selected *SubstructureRedirectMask* on the parent, then a *ConfigureRequest* event is generated, and no further processing is performed.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XRestackWindows — change stacking order of siblings.

SYNOPSIS

```
XRestackWindows(display, windows, nwindows);
  Display *display;
  Window windows[];
  int nwindows;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
windows	Specifies an array containing the windows to be restacked. All the windows must have a common parent.
nwindows	Specifies the number of windows to be restacked.

DESCRIPTION

XRestackWindows restacks the windows in the order specified, from top to bottom. The stacking order of the first window in the *windows* array will be on top, and the other windows will be stacked underneath it in the order of the array. Note that other siblings may not be included in the *windows* array and so the top window in that array will not move relative to these other siblings.

If the *override_redirect* attribute of the window is *False* and some other client has selected *SubstructureRedirectMask* on the parent, then *ConfigureRequest* events are generated for each window whose *override_redirect* is not set, and no further processing is performed. Otherwise, the windows will be restacked in top to bottom order.

ERRORS

BadWindow

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XRotateBuffers — rotate the cut buffers.

SYNOPSIS

```
XRotateBuffers(display, rotate)
  Display *display;
  int rotate;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
rotate	Specifies how many positions to rotate the cut buffers.

DESCRIPTION

XRotateBuffers rotates the 8 cut buffers the amount specified by *rotate*. Buffer 0 becomes buffer *rotate*, buffer 1 becomes buffer *rotate*+1 mod 8, buffer 2 becomes buffer *rotate*+2 mod 8, and so on. This cut buffer numbering is global to the display. This routine will not work if any of the buffers have not been stored into with *XStoreBuffer*.

Refer to the *GSE Programmer's Guide* for a description of cut buffers.

ERRORS

BadAtom
BadMatch
BadWindow

SEE ALSO

XStoreBuffer, XStoreBytes, XFetchBuffer, XFetchBytes.

NAME

XRotateWindowProperties — rotate properties in the properties array.

SYNOPSIS

XRotateWindowProperties (*display*, *w*, *properties*, *num_prop*, *npositions*)
  Display **display*;
  Window *w*;
  Atom *properties*[] ;
  int *num_prop*;
  int *npositions*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
properties	Specifies the property list.
num_prop	Specifies the length of the properties array.
npositions	Specifies the number of positions to rotate property list. The sign controls the direction of rotation.

DESCRIPTION

XRotateWindowProperties rotates the contents of an array of properties on a window. If the property names in the *properties* array are viewed as being numbered starting from zero and if there are *num_prop* property names in the list, then the value associated with property name I becomes the value associated with property name (I + *npositions*) mod *num_prop*, for all I from zero to *num_prop* - 1. Therefore, the sign of *npositions* controls the direction of rotation. The effect is to rotate the states by *npositions* places around the virtual ring of property names (right for positive *npositions*, left for negative *nposition*).

If *npositions* mod *num_prop* is non-zero, a *PropertyNotify* event is generated for each property, in the order listed.

If a *BadAtom* or *BadMatch* error is generated, no properties are changed.

ERRORS

BadAtom	Atom occurs more than once in list for the window. No property with that name for the window.
BadMatch	
BadWindow	

SEE ALSO

XSetStandardProperties, XGetFontProperty, XDeleteProperty, XChangeProperty, XGetWindowProperty, XListProperties, XGetAtomName, XInternAtom.

NAME

XSaveContext — save data value corresponding to window and context type (not graphics context).

SYNOPSIS

```
int XSaveContext(display, w, context, data)
  Display *display;
  Window w;
  XContext context;
  caddr_t data;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window with which the data is associated.
context	Specifies the context type to which the data corresponds.
data	Specifies the data to be associated with the window and type.

DESCRIPTION

XSaveContext saves *data* to the context manager database, according to the specified *window* and *context* ID. The context manager is used for associating data with windows within an application. The client must have called *XUniqueContext* to get the *context* ID before calling this function. The meaning of the *data* is indicated by the *context* ID, but is completely up to the client.

If an entry with the specified *window* and *context* ID already exists, *XSaveContext* writes over it with the specified data. However, this has costs in time and space. If you know the entry already exists, it is better to call *XDeleteContext* first.

The *XSaveContext* function returns *XCNOMEM* (a nonzero error code) if an error has occurred and zero (0) otherwise. Refer to the *GSE Programmer's Guide* for a description of a context manager.

STRUCTURES

```
typedef int XContext;
```

SEE ALSO

XDeleteContext, XFindContext, XUniqueContext.

NAME

XSelectInput — select the event types to be sent to a window.

SYNOPSIS

```
XSelectInput(display, w, event_mask)
  Display *display;
  Window w;
  long event_mask;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window interested in the input events.
event_mask	Specifies the event mask. This mask is the bitwise OR of one or more of the valid event mask bits (see below).

DESCRIPTION

XSelectInput defines which input events the window is interested in. If a window is not interested in an event, it propagates up to the closest ancestor unless otherwise specified in the *do_not_propagate_mask* attribute.

The bits of the mask are defined in <*X11/X.h*> :

ButtonPressMask	*NoEventMask*
ButtonReleaseMask	*KeyPressMask*
EnterWindowMask	*KeyReleaseMask*
LeaveWindowMask	*ExposureMask*
PointerMotionMask	*VisibilityChangeMask*
PointerMotionHintMask	*StructureNotifyMask*
Button1MotionMask	*ResizeRedirectMask*
Button2MotionMask	*SubstructureNotifyMask*
Button3MotionMask	*SubstructureRedirectMask*
Button4MotionMask	*FocusChangeMask*
Button5MotionMask	*PropertyChangeMask*
ButtonMotionMask	*ColormapChangeMask*
KeyMapStateMask	*OwnerGrabButtonMask*

A call on *XSelectInput* overrides any previous call on *XSelectInput* for the same window from the same client but not for other clients. Multiple clients can select input on the same window; their *event_masks* are disjoint. When an event is generated it will be reported to all interested clients. However, only one client at a time can select for each of *SubstructureRedirectMask, ResizeRedirectMask,* and *ButtonPress.*

If a window has both *ButtonPressMask* and *ButtonReleaseMask* selected, then a *ButtonPress* event in that window will automatically grab the mouse until all buttons are released, with events sent to windows as described for *XGrabPointer.* This ensures that a window will see the *ButtonRelease* event corresponding to the *ButtonPress* event, even though the mouse may have exited the window in the meantime.

If *PointerMotionMask* is selected, events will be sent independent of the state of the mouse buttons. If instead, one or more of *Button1MotionMask, Button2MotionMask, Button3MotionMask, Button4MotionMask, Button5MotionMask* is selected, *MotionNotify* events will be generated only when one or more of the specified buttons is depressed.

XOpenDisplay sets the *event_mask* attribute; this attribute can also be set directly with *XChangeWindowAttributes.*

ERRORS

BadValue
BadWindow

SEE ALSO

XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XSendEvent — send an event.

SYNOPSIS

```
Status XSendEvent(display, w, propagate, event_mask, event)
  Display *display;
  Window w;
  Bool propagate;
  long event_mask;
  XEvent *event;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
w	Specifies the window ID of the window where you want to send the event. Pass the window resource ID, *PointerWindow,* or *InputFocus.*
propagate	Specifies how the sent event should propagate depending on *event_mask.* See description below. May be *True* or *False.*
event_mask	Specifies the event mask. See *XSelectInput* for a detailed list of the Event masks.
event	Specifies a pointer to the event to be sent.

DESCRIPTION

XSendEvent sends an event from one client to another (or conceivably to itself). This function is used for communication between clients using selections, for simulating user actions in demos, and more.

The specified event is sent to the window indicated by *w* regardless of active grabs.

If *w* is set to *PointerWindow,* the destination of the event will be the window that the pointer is in. If *w* is *InputFocus* is specified, then the destination is the focus window, regardless of pointer position.

If *propagate* is *False,* then the event is sent to every client selecting on the window specified by *w* any of the event types in *event_mask.* If *propagate* is *True* and no clients have been selected on *w* any of the event types in *event_mask,* then the event propagates like any other event.

The event code must be one of the core events, or one of the events defined by a loaded extension, so that the server can correctly byte swap the contents as necessary. The contents of the event are otherwise unaltered and unchecked by the server except that in Release 1 the most significant bit of *XEvent.type* is set to 1. In Release 2, the high bit is no longer set. Instead, a new flag *send_event* has been added to each event, which if *True* indicates that the event was sent with *XSendEvent*.

Under Release 1, if a client wants to read events sent by *XSendEvent* as normal events, it must ignore the high bit by ORing the event type with the following expression:

```
XEvent report;
XNextEvent(display, &report);
report.type &= 0x7f;
/* now sent event looks like any other */
```

This function is often used in selection processing. For example, the owner of a selection should use *XSendEvent* to send a *SelectionNotify* event to a requestor when a selection has been converted and stored as a property.

STRUCTURES

Refer to the *GSE Programmer's Guide.*

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, QLength.

NAME

XSetAccessControl — disable or enable access control.

SYNOPSIS

```
XSetAccessControl(display, mode)
  Display  *display;
  int mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
mode	Specifies whether you want to change the access control to enable or disable. Pass one of these constants: *EnableAccess* or *DisableAccess*.

DESCRIPTION

XSetAccessControl specifies whether other applications (running on the current host) that subsequently connect to the server should be able to modify the host access list.

ERRORS

BadAccess
BadAlloc

SEE ALSO

XAddHost, XAddHosts, XListHosts, XRemoveHost, XRemoveHosts, XDisableAccessControl, XEnableAccessControl.

NAME

XSetAfterFunction — set function called after all Xlib functions.

SYNOPSIS

```
int (*XSetAfterFunction(display, func))()
  Display *display;
  int (*func)();
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
func	Specifies the user-defined function to be called after each Xlib function. This function is called with one argument, the *display* pointer.

DESCRIPTION

All Xlib functions that generate protocol requests call what is known as an "after function" after completing their work. *XSetAfterFunction* sets the function to be called.

SEE ALSO

XDisplayName, XGetErrorDatabaseText, XGetErrorText, XSetErrorHandler, XSetIOErrorHandler, XSynchronize.

NAME

XSetArcMode — set arc mode in graphics context.

SYNOPSIS

```
XSetArcMode(display, gc, arc_mode)
  Display *display;
  GC gc;
  int arc_mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
arc_mode	Specifies the arc mode for the specified graphics context. Possible values are *ArcChord* or *ArcPieSlice*.

DESCRIPTION

XSetArcMode sets the *arc_mode* member of the GC, which controls filling in the *XFillArcs* function. *ArcChord* specifies that the area between the arc and a line segment joining the end points of the arc is filled. *ArcPieSlice* specifies that the area filled is delimited by the arc and two line segments connecting the ends of the arc to the center point of the rectangle defining the arc.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetBackground — set background pixel value in graphics context.

SYNOPSIS

```
XSetBackground(display, gc, background)
  Display *display;
  GC gc;
  unsigned long background;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
background	Specifies the background you want to set for the specified graphics context.

DESCRIPTION

XSetBackground sets the *background* pixel value for graphics requests. Note that this is different from the *background* of a window, which can be set with either *XSetWindowBackground* or *XSetWindowBackgroundPixmap*.

ERRORS

BadGC

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetClassHint — set WM_CLASS property of window.

SYNOPSIS

```
XSetClassHint(display, w, class_hints)
  Display  *display;
  Window w;
  XClassHint  *class_hints;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
class_hints	Specifies the *XClassHint* structure that is to be used.

DESCRIPTION

XSetClassHint sets the WM_CLASS property for the specified window.

XSetClassHint returns a status of 0 on failure, non-zero on success.

The *XClassHint* structure set contains *res_class,* which is the name of the client such as "emacs", and *res_name,* which is the first of the following that applies:

- command line option (**–rn** *name*)
- a specific environment variable (e.g., RESOURCE_NAME)
- the trailing component of `argv[0]`

STRUCTURES

```
typedef struct {
  char *res_name;
  char *res_class;
} XClassHint;
```

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XSetClipMask — set clip_mask pixmap in graphics context.

SYNOPSIS

```
XSetClipMask(display, gc, pixmap)
  Display *display;
  GC gc;
  Pixmap pixmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
pixmap	Specifies the pixmap. Pass the constant *None* if no clipping is desired.

DESCRIPTION

XSetClipMask sets the *clip_mask* member of a GC. The *clip_mask* filters which pixels in the destination are drawn. Use *XSetClipRectangles* to set the *clip_mask* to a set of rectangles, or *XSetRegion* to set the *clip_mask* to a region.

ERRORS

BadMatch
BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetClipOrigin — set clip origin in graphics context.

SYNOPSIS

XSetClipOrigin(*display*, *gc*, *clip_x_origin*, *clip_y_origin*)
  Display **display*;
  GC *gc*;
  int *clip_x_origin*, *clip_y_origin*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
clip_x_origin *clip_y_origin*	Specify the clip origin relative to the window specified in the GC.

DESCRIPTION

XSetClipOrigin sets the *clip_x_origin* and *clip_y_origin* members of the GC. The clip origin control the position of the *clip_mask,* which filters which pixels in the destination are drawn.

ERRORS

BadGC

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetClipRectangles — change clip_mask in graphics context to list of rectangles.

SYNOPSIS

XSetClipRectangles (*display*, *gc*, *clip_x_origin*, *clip_y_origin*, *rectangles*, *nrects*, *ordering*)
  Display **display*;
  GC *gc*;
  int *clip_x_origin*, *clip_y_origin*;
  XRectangle *rectangles*[];
  int *nrects*;
  int *ordering*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
clip_x_origin *clip_y_origin*	Specify the x and y coordinates of the clip origin, relative to the window specified in the drawing request.
rectangles	Specifies an array of rectangles. These are the rectangles you want output clipped to.
nrects	Specifies the number of rectangles.
ordering	Specifies the ordering relations on the rectangles. Possible values are *Unsorted, YSorted, YXSorted,* or *YXBanded.*

DESCRIPTION

XSetClipRectangles changes the *clip_mask* in the specified GC to the specified list of rectangles and sets the clip origin to *clip_x_origin* and *clip_y_origin*. The rectangle coordinates are interpreted relative to the clip origin. The output from drawing requests using that GC are henceforth clipped to remain contained within the rectangles. The rectangles should be nonintersecting, or the graphics results will be undefined. If the list of rectangles is empty, output is effectively disabled as all space is clipped in that GC. This is the opposite of a *clip_mask* of *None* in *XCreateGC, XChangeGC,* or *XSetClipMask.*

If known by the client, ordering relations on the rectangles can be specified with the *ordering* argument. This may provide faster operation by the server. If an incorrect ordering is specified, the X server may generate a

BadMatch error, but it is not required to do so. If no error is generated, the graphics results are undefined. *Unsorted* means the rectangles are in arbitrary order. *YSorted* means that the rectangles are nondecreasing in their Y origin. *YXSorted* additionally constrains *YSorted* order in that all rectangles with an equal Y origin are nondecreasing in their X origin. *YXBanded* additionally constrains *YXSorted* by requiring that, for every possible horizontal Y scan line, all rectangles that include that scan line have identical Y origins and Y extents.

To cancel the effect of this command, so that there is no clipping, pass *None* as the *clip_mask* in *XChangeGC* or *XSetClipMask*.

STRUCTURES

```
typedef struct {
  short x,y;
  unsigned short width, height;
} XRectangle;
```

ERRORS

BadAlloc	
BadGC	
BadMatch	Incorrect ordering (error message server-dependent).
BadValue	

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetCloseDownMode — change close down mode of client.

SYNOPSIS

```
XSetCloseDownMode(display, close_mode)
  Display *display;
  int close_mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
close_mode	Specifies the client close down mode you want. Pass one of these constants: *DestroyAll*, *RetainPermanent*, or *Retain-Temporary*.

DESCRIPTION

XSetCloseDownMode defines what will happen to the client's resources at connection close. A connection between a client and the server starts in *DestroyAll* mode, and all resources associated with that connection will be freed when the client process dies. If the close down mode is *RetainTemporary* or *RetainPermanent* when the client dies, its resources live on until a call to *XKillClient*. The *resource* argument of *XKillClient* can be used to specify which client to kill, or it may be the constant *AllTemporary*, in which case *XKillClient* kills all resources of all clients that have terminated in *RetainTemporary* mode.

ERRORS

BadValue

SEE ALSO

XKillClient

NAME

XSetCommand — set the WM_COMMAND atom (command line args).

SYNOPSIS

```
XSetCommand(display, w, argv, argc)
  Display *display;
  Window w;
  char **argv;
  int argc;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
argv	Specifies a pointer to the command and arguments used to start the application.
argc	Specifies the number of arguments.

DESCRIPTION

XSetCommand is used by the application to set the WM_COMMAND property for the window manager with the SYSTEM V/68 shell command and its arguments used to invoke the application.

Use this command only if not calling *XSetStandardProperties*.

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes.

NAME

XSetDashes — set *dash_offset* and *dash_list* (for lines) of graphics context.

SYNOPSIS

```
XSetDashes(display, gc, dash_offset, dash_list, n)
  Display *display;
  GC gc;
  int dash_offset;
  char dash_list[] ;
  int n;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
dash_offset	Specifies the phase of the pattern for the dashed line style.
dash_list	Specifies the dash list for the dashed line style. An odd-length list is equivalent to the same list concatenated with itself to produce an even-length list.
n	Specifies the length of the dash list argument.

DESCRIPTION

XSetDashes sets the *dashes* member of the GC. The initial and alternating elements of the *dash_list* are the "even" dashes, the others are the "odd" dashes. All of the elements must be non-zero. The *dash_offset* defines the phase of the pattern, specifying how many elements into the *dash_list* the pattern should actually begin in the line drawn by the request.

n specifies the length of *dash_list*. An odd value for *n* is interpreted as specifying the *dash_list* concatenated with itself to produce twice as long a list.

The unit of measure for dashes is the same as in the ordinary coordinate system. Ideally, a dash length is measured along the slope of the line, but server implementors are only required to match this ideal for horizontal and vertical lines. Failing the ideal semantics, it is suggested that the length be measured along the major axis of the line. The major axis is defined as the x axis for lines drawn at an angle of between –45 and +45 degrees or between 315 and 225 degrees from the x axis. For all other lines, the major axis is the y axis.

The default *dash_list* in a newly created GC is equivalent to [4,4].

Refer to the *GSE Programmer's Guide* for further information.

ERRORS

BadAlloc	
BadGC	
BadValue	No values in *dash_list*. Element in *dash_list* is zero.

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetErrorHandler — set non-fatal error event handler.

SYNOPSIS

XSetErrorHandler (*handler*)
  int (* *handler*) **(Display *, XErrorEvent *)**

ARGUMENTS

handler	The user-defined function to be called to handle error events. If a NULL pointer, reinvoke the default handler, which prints a message and exits.

DESCRIPTION

The error handler function specified in *handler* will be called by Xlib whenever an *XError* event is received. These are non-fatal conditions, such as unexpected values for arguments. It is acceptable for this procedure to return, though the default handler simply prints a message and exits. However, the error handler should NOT perform any operations (directly or indirectly) on the *Display*.

The function is called with two arguments, the display variable and a pointer to the *XErrorEvent* structure. Here is a trivial example of a user-defined error handler:

```
int myhandler (display, myerr)
Display *display;
XErrorEvent *myerr;
{
char msg[80];
XGetErrorText(display, myerr->error_code, msg, 80);
fprintf(stderr, "Error code %s\n", msg);
}
```

This is how the example routine would be used in *XSetErrorHandler.*

```
XSetErrorHandler(myhandler);
```

Note that *XSetErrorHandler* is one of the few routines that does not require a display argument. The routine that calls the error handler gets the display variable from the *XErrorEvent* structure.

The error handler is not called on *BadName* errors from *OpenFont, Lookup-Color, AllocNamedColor,* protocol requests, on *BadFont* errors from a

QueryFont protocol request, or on *BadAlloc* or *BadAccess* errors. These errors can be caught and handled by the program by checking the return value of the routine.

Use *XIOErrorHandler* to to provide a handler for fatal errors.

In the *XErrorEvent* structure shown below, The *serial* member is the number of requests starting from one sent over the network connection since it was opened. It is the number that was the value of the request sequence number immediately after the failing call was made. The *request_code* member is a protocol representation of the name of the procedure that failed and are defined in < *X11/X.h* >.

STRUCTURES

```
typedef struct {
        int type
        Display *display;       /* Display the event was read from */
        unsigned long serial;   /* serial number of failed request */
        char error_code;        /* error code of failed request */
        char request_code;      /* Major op-code of failed request */
        char minor_code;        /* Minor op-code of failed request */
        XID resourceid;         /* resource id */
} XErrorEvent;
```

SEE ALSO

XDisplayName, XGetErrorDatabaseText, XGetErrorText, XSetIOErrorHandler, XSynchronize, XSetAfterFunction.

NAME

XSetFillRule — set fill rule in graphics context.

SYNOPSIS

```
XSetFillRule(display, gc, fill_rule)
  Display *display;
  GC gc;
  int fill_rule;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
gc	Specifies the graphics context.
fill_rule	Specifies the fill rule you want to set for the specified graphics context. Possible values are *EvenOddRule* or *WindingRule.*

DESCRIPTION

XSetFillRule sets the *fill_rule* member of a GC. The *fill_rule* member of the GC determines what pixels are drawn in *XFillPolygon* requests. Simply put, *WindingRule* fills overlapping areas of the polygon, while *EvenOddRule* does not fill areas that overlap an odd number of times. Technically, *EvenOddRule* means that the point is drawn if an arbitrary ray drawn from the point would cross the path determined by the request an odd number of times. *WindingRule* indicates that a point is drawn if a point crosses an unequal number of clockwise and counterclockwise path segments, as seen from the point.

A clockwise directed path segment is one which crosses the ray from left to right as observed from the point. A counterclockwise segment is one which crosses the ray from right to left as observed from the point. The case where a directed line segment is coincident with the ray is uninteresting because you can simply choose a different ray that is not coincident with a segment.

All calculations are performed on infinitely small points, so that if any point within a pixel is considered inside, the entire pixel is drawn. Pixels with centers exactly on boundaries are considered inside only if the filled area is to the right, except that on horizontal boundaries, the pixel is considered inside only if the filled area is below the pixel.

Refer to the *GSE Programmer's Guide* for more information.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetFillStyle — set fill style in graphics context.

SYNOPSIS

```
XSetFillStyle(display, gc, fill_style)
  Display  *display;
  GC gc;
  int fill_style;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
fill_style	Specifies the fill style for the specified graphics context. Possible values are *FillSolid, FillTiled, FillStippled,* or *FillOpaqueStippled.*

DESCRIPTION

XSetFillStyle sets the *fill_style* member of the GC. The *fill_style* defines the contents of the source for line, text, and fill requests. *FillSolid* indicates that the pixels represented by set bits in the source are drawn in the *foreground* pixel value, and unset bits in the source are not drawn. *FillTiled* uses the *tile* specified in the GC to determine the pixel values for set bits in the source. *FillOpaqueStippled* specifies that bits set in the *stipple* are drawn in the *foreground* pixel value and unset bits are drawn in the *background.* *FillStippled* draws bits set in the source and set in the stipple in the *foreground* color, and leaves unset bits alone.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetFont — set current font in graphics context.

SYNOPSIS

```
XSetFont(display, gc, font)
  Display  *display;
  GC gc;
  Font font;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
font	Specifies the font ID.

DESCRIPTION

XSetFont sets the *font* in the GC. Text drawing requests using this GC will use this font only if it is loaded. Otherwise, the text will not be drawn.

ERRORS

BadAlloc
BadFont
BadGC

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFontPath, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XSetFontPath — set the font search path.

SYNOPSIS

XSetFontPath(*display*, *directories*, *ndirs*)
  Display **display*;
  char ***directories*;
  int *ndirs*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
directories	Specifies the directory path used to look for the font. Setting the path to the empty list restores the default path defined for the X server.
ndirs	Specifies the number of directories in the path.

DESCRIPTION

XSetFontPath defines the directory search path for font lookup for all clients. Therefore the user should construct a new directory search path carefully by adding to the old directory search path obtained by *XGetFontPath*. Passing an invalid path can result in preventing the server from accessing any fonts. Also avoid restoring the default path, since some other client may have changed the path on purpose.

The interpretation of the strings is operating system dependent, but they are intended to specify directories to be searched in the order listed. Also, the contents of these strings are operating system specific and are not intended to be used by client applications.

As a side-effect of executing this request, the server is guaranteed to flush all cached information about fonts for which there are currently no explicit resource IDs allocated. The meaning of errors from this request is system specific.

ERRORS

BadValue

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XUnloadFont, XGetFontProperty, XCreateFontCursor.

NAME

XSetForeground — set foreground pixel value in graphics context.

SYNOPSIS

```
XSetForeground(display, gc, foreground)
  Display *display;
  GC gc;
  unsigned long foreground;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
foreground	Specifies the foreground pixel value you want for the specified graphics context.

DESCRIPTION

XSetForeground sets the *foreground* member in a GC. This pixel value is used for set bits in the source according to the *fill_style*.

Refer to the *GSE Programmer's Guide* for more information on the GC.

ERRORS

BadGC

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetFunction — set bitwise logical operation in graphics context.

SYNOPSIS

```
XSetFunction(display, gc, function)
  Display *display;
  GC gc;
  int function;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
function	Specifies the logical operation you want for the specified graphics context. See Description for the choices and their meanings.

DESCRIPTION

XSetFunction sets the logical operation applied between the source pixel values (generated by the drawing request) and existing destination pixel values (already in the window or pixmap) to generate the final destination pixel values in a drawing request (what is actually drawn to the window or pixmap). Of course, the *plane_mask* and *clip_mask* in the GC also affect this operation by preventing drawing to planes and pixels respectively.

Refer to the *GSE Programmer's Guide* for more information about the logical function.

The *function* symbols and their logical definitions are:

Symbol	Bit	Meaning
GXclear	0x0	0
GXand	0x1	src AND dst
GXandReverse	0x2	src AND (NOT dst)
GXcopy	0x3	src
GXandInverted	0x4	(NOT src) AND dst
GXnoop	0x5	dst
GXxor	0x6	src XOR dst
GXor	0x7	src OR dst
GXnor	0x8	(NOT src) AND (NOT dst)
GXequiv	0x9	(NOT src) XOR dst
GXinvert	0xa	(NOT dst)
GXorReverse	0xb	src OR (NOT dst)
GXcopyInverted	0xc	(NOT src)
GXorInverted	0xd	(NOT src) OR dst
GXnand	0xe	(NOT src) OR (NOT dst)
GXset	0xf	1

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetGraphicsExposures — set graphics-exposures flag in graphics context.

SYNOPSIS

```
XSetGraphicsExposures(display, gc, graphics_exposures)
  Display *display;
  GC gc;
  Bool graphics_exposures;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
graphics_exposures	Specifies whether you want *GraphicsExpose* and *NoExpose* events when calling *XCopyArea* and *XCopyPlane* with this graphics context.

DESCRIPTION

XSetGraphicsExposure sets the *graphics_exposures* member of the GC. If *graphics_exposures* is *True, GraphicsExpose* events will be generated when *XCopyArea* and *XCopyPlane* requests cannot be completely satisfied because a source region is obscured; and *NoExpose* events are generated when they can be completely satisfied. If *graphics_exposures* is *False,* these events are not generated.

These events are not selected in the normal way with *XSelectInput*. Setting the *graphics_exposures* member of the GC used in the *CopyArea* or *CopyPlane* request is the only way to select these events.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetIconName — set name to be displayed in a window's icon.

SYNOPSIS

```
XSetIconName(display, w, icon_name)
  Display *display;
  Window w;
  char *icon_name;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose icon name is being set.
icon_name	Specifies the name to be displayed in the window's icon. The name should be a null-terminated string. This name is returned by any subsequent call to *XGetIconName*.

DESCRIPTION

XSetIconName sets the WM_ICON_NAME property for a window. This is usually set by an application for the window manager. This should be set to a short name to be displayed in association with an icon.

XSetStandardProperties also sets this property.

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XSetIconSizes — set value of the WM_ICON_SIZE property.

SYNOPSIS

```
XSetIconSizes(display, w, size_list, count)
  Display *display;
  Window w;
  XIconSize *size_list;
  int count;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
size_list	Specifies a pointer to the size list.
count	Specifies the number of items in the size list.

DESCRIPTION

XSetIconSizes is normally used by a window manager to set the range of preferred icon sizes in the WM_ICON_SIZE property of the root window.

Applications can then read the property with *XGetIconSizes.*

STRUCTURES

```
typedef struct {
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
} XIconSize;
```

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetCommand.

NAME

XSetInputFocus — set the input focus window.

SYNOPSIS

```
XSetInputFocus(display, focus, revert_to, time)
  Display *display;
  Window focus;
  int revert_to;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
focus	Specifies the window ID of the window you want to be the input focus. Pass the window ID, *PointerRoot*, or *None*.
revert_to	Specifies which window the input focus reverts to if the focus window becomes not viewable. Pass one of these constants: *RevertToParent*, *RevertToPointerRoot*, or *RevertTo-None*. Must not be a window ID.
time	Specifies the time when the focus change should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime*. Also returns the time of the focus change when *CurrentTime* is specified.

DESCRIPTION

XSetInputFocus changes the input focus and the last-focus-change time. The function has no effect if *time* is earlier than the current last-focus-change time or later than the current X server time. Otherwise, the last-focus-change time is set to the specified time, with *CurrentTime* replaced by the current X server time.

XSetInputFocus generates *FocusIn* and *FocusOut* events if *focus* is different from the current focus.

XSetInputFocus executes as follows, depending on what value you assign to the *focus* argument:

- If you assign *None,* all keyboard events are discarded until you set a new focus window. In this case, *revert_to* is ignored.
- If you assign a window ID, it becomes the main keyboard's focus window. If a generated keyboard event would normally be reported to this window or one of its inferiors, the event is reported normally; otherwise, the event is reported with respect to the focus window.
- If you assign *PointerRoot,* the focus window is dynamically taken to be the root window of whatever screen the pointer is on at each keyboard event. In this case, *revert_to* is ignored.

The specified focus window must be viewable at the time of the request (else a *BadMatch* error). If the focus window later becomes not viewable, the focus window will change to the *revert_to* argument.

If the focus window later becomes not viewable, *XSetInputFocus* evaluates the *revert_to* argument to determine the new focus window:

- If you assign *RevertToParent,* the focus reverts to the parent (or the closest viewable ancestor) automatically with a new *revert_to* argument of *RevertToName.*
- If you assign *RevertToPointerRoot* or *RevertToNone,* the focus reverts to that value automatically. *FocusIn* and *FocusOut* events are generated when the focus reverts, but the *last_focus_change_time* is not affected.

ERRORS

BadMatch	*focus* window not viewable when *XSetInput* called.
BadValue	
BadWindow	

SEE ALSO

XSelectInput, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XSetIOErrorHandler — handle fatal I/O errors.

SYNOPSIS

```
XSetIOErrorHandler(handler)
  int (*handler)(Display *);
```

ARGUMENTS

handler	Specifies a pointer to a user-defined fatal error handling routine. If *NULL,* reinvoke the default fatal error handler.

DESCRIPTION

XSetIOErrorHandler specifies a user-defined error handling routine for fatal errors. This error handler will be called by Xlib if any sort of system call error occurs, such as the connection to the server being lost. The called routine should not return. If the I/O error handler does return, the client process will exit.

If *handler* is a *NULL* pointer, the default error handler is reinvoked. The default I/O error handler prints an error message and exits.

SEE ALSO

XDisplayName, XGetErrorDatabaseText, XGetErrorText, XSetErrorHandler, XSynchronize, XSetAfterFunction.

NAME

XSetLineAttributes — set line drawing components in graphics context.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
line_width	Specifies the line width you want to set for the specified graphics context.
line_style	Specifies the line style you want to set for the specified graphics context. Possible values are *LineSolid, LineOnOff-Dash,* or *LineDoubleDash.*
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.*
join_style	Specifies the line-join style you want to set for the specified graphics context. Possible values are *JoinMiter, Join-Round,* or *JoinBevel.*

DESCRIPTION

XSetLineAttributes sets four types of line characteristics in the GC: *line_width, line_style, cap_style,* and *join_style.*

Refer to the description of line and join styles in the *GSE Programmer's Guide.* See also *XSetDashes.*

A *line_width* of zero (0) means use the fastest algorithm for drawing a line of one-pixel width. These lines may not meet properly with lines specified as width 1 or more.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetModifierMapping — set keycodes to be used as modifiers (Shift, Control, etc.).

SYNOPSIS

```
int XSetModifierMapping(display, mod_map)
  Display *display;
  XModifierKeymap *mod_map;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
mod_map	Specifies a pointer to the *XModifierKeymap* structure.

DESCRIPTION

XSetModifierMapping is one of two ways to specify the keycodes of the keys that are to be used as modifiers (like Shift, Control, etc.). *XSetModifierMapping* specifies all the keycodes for all the modifiers at once. The other, easier, way is to use *XInsertModifiermapEntry* and *XDeleteModifiermapEntry* which add or delete a single keycode for a single modifier. *XSetModifierMapping* does the work in a single call, but the price of this call is that you need to manually set up the *XModifierKeymap* structure pointed to by *mod_map*. This requires you to know how the *XModifierKeymap* structure is defined and organized, as described in the next three paragraphs.

The *XModifierKeymap* structure for the *mod_map* argument should be created using *XNewModifierMap* or *XGetModifierMapping*. The *max_keypermod* element of the structure specifies the maximum number of keycodes that can be mapped to each modifier. You define this number, but there may be an upper limit on a particular server.

The *modifiermap* element of the structure is an array of keycodes. There are eight by *max_keypermod* keycodes in this array: eight because there are eight modifiers, and *max_keypermod* because that is the number of keycodes that must be reserved for each modifier.

The eight modifiers are represented by the constants *ShiftMapIndex*, *LockMapIndex*, *ControlMapIndex*, *Mod1MapIndex*, *Mod2MapIndex*, *Mod3MapIndex*, *Mod4MapIndex*, and *Mod5MapIndex*. These are not actually used as arguments, but they are convenient for referring to each row in the *modifiermap* structure while filling it. The definitions of these constants are shown in the Structures section below.

Now you can interpret the *modifiermap* array. For each modifier in a given *modifiermap,* the keycodes which correspond are from `modifiermap[index * max_keypermod]` to `modifiermap[[(index + 1) * max_keyspermod] -1]` where *index* is the appropriate modifier index definition (*ShiftMapIndex, LockMapIndex,* etc.). You must set the *mod_map* array up properly before calling *XSetModifierMapping.* Now you know why *XInsertModifierMapEntry* and *XDeleteModifierMapEntry* were created!

Zero keycodes are ignored. No keycode may appear twice anywhere in the map (otherwise, a *BadValue* error is generated). In addition, all of the non-zero keycodes must be in the range specified by *min_keycode* and *max_keycode* in the *Display* structure (else a *BadValue* error).

A server can impose restrictions on how modifiers can be changed. For example, certain keys may not generate up transitions in hardware, or multiple modifier keys may not be supported. If a restriction is violated, then the status reply is *MappingFailed,* and none of the modifiers are changed.

If the new keycodes specified for a modifier differ from those currently defined and any (current or new) keys for that modifier are in the down state, then the status reply is *MappingBusy,* and none of the modifiers are changed.

XSetModifierMapping generates a *MappingNotify* event on a *MappingSuccess* status.

A zero value for *modifiermap* indicates that no keys are valid as any modifier.

STRUCTURES

```
typedef struct {
  int max_keypermod;      /* server's max # of keys per modifier */
  KeyCode *modifiermap;   /* an 8 by max_keypermod array */
} XModifierKeymap;

/* modifier names.  Used to build a SetModifierMapping request or
   to read a GetModifierMapping request.  These correspond to the
   masks defined above. */
#define ShiftMapIndex          0
#define LockMapIndex           1
#define ControlMapIndex        2
#define Mod1MapIndex           3
#define Mod2MapIndex           4
#define Mod3MapIndex           5
#define Mod4MapIndex           6
#define Mod5MapIndex           7
```

ERRORS

BadAlloc	
BadValue	Keycode appears twice in the map. Keycode < *display->min_keycode* or keycode > *display->max_keycode.*

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XStringToKeysym, XLookupKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XGetModifierMapping, XInsertModifiermapEntry, XDeleteModifiermapEntry.

NAME

XSetNormalHints — set size hints for window in normal state (not zoomed).

SYNOPSIS

```
void XSetNormalHints(display, w, hints)
  Display *display;
  Window w;
  XSizeHints *hints;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
hints	Specifies a pointer to the sizing hints for the window in its normal state.

DESCRIPTION

XSetNormalHints sets the WM_NORMAL_HINTS property for the specified window. Applications use *XSetNormalHints* to inform the window manager of the size or position desirable for that window. In addition, an application wanting to move or resize itself should call *XSetNormalHints* specifying its new desired location and size, instead of making direct X calls to move or resize. This is because some window managers may redirect window configuration requests, but ignore the resulting events and pay attention to property changes instead.

To set size hints, an application must not only assign values to the appropriate elements in the hints structure, but also must set the *flags* field of the structure to indicate which members have assigned values and where it came from.

STRUCTURES

```
typedef struct {
  long flags;          /* which fields in structure are defined */
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
         int x;              /* numerator */
         int y;              /* denominator */
  } min_aspect, max_aspect;
} XSizeHints;

/* flags argument in size hints */
#define USPosition (1L << 0) /* user specified x, y */
#define USSize     (1L << 1) /* user specified width, height */

#define PPosition  (1L << 2) /* program specified position */
#define PSize      (1L << 3) /* program specified size */
#define PMinSize   (1L << 4) /* program specified minimum size */
#define PMaxSize   (1L << 5) /* program specified maximum size */
#define PResizeInc (1L << 6) /* program specified resize increments */
#define PAspect    (1L << 7) /* program specified min/max aspect ratios */
#define PAllHints (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
```

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XSetPlaneMask — set plane mask in graphics context.

SYNOPSIS

```
XSetPlaneMask(display, gc, plane_mask)
  Display *display;
  GC gc;
  unsigned long plane_mask;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
plane_mask	Specifies the plane mask. You can use the macro *AllPlanes* if desired.

DESCRIPTION

XSetPlaneMask sets the *plane_mask* member of the specified GC. The *plane_mask* determines which planes of the destination drawable are affected by a graphics request.

ERRORS

BadGC

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetPointerMapping — set pointer button mapping.

SYNOPSIS

```
int XSetPointerMapping(display, map, nmap)
  Display *display;
  unsigned char map[];
  int nmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
map	Specifies the mapping list.
nmap	Specifies the number of items in the mapping list.

DESCRIPTION

XSetPointerMapping sets the mapping of the pointer. Elements of the *map* list are indexed starting from one. The length of the list *nmap* must be the same as *XGetPointerMapping* returns (you must call that first). The index is a physical button number, and the element of the list defines the effective button number. In other words, if *map[2]* is set to *1*, when the second physical button is pressed, a *ButtonPress* event will be generated if *Button1Mask* was selected but not if *Button2Mask* was selected. The *button* member in the event will read *Button1*.

No two elements can have the same nonzero value. A zero value for an element of *map* disables a button, and values for elements are not restricted in value by the number of physical buttons. If any of the buttons to be altered are currently in the down state, the status reply is *MappingBusy* and the mapping is not changed.

This function returns either *MappingSuccess* or *MappingBusy*. *XSetPointerMapping* generates a *MappingNotify* event on a status of *MappingSuccess*.

ERRORS

BadValue	Two elements of *map*[] have same non-zero value. *nmap* not equal to *XGetPointerMapping* return value.

SEE ALSO

XQueryPointer, XWarpPointer, XGrabPointer, XChangeActivePointerGrab, XUngrabPointer, XGetPointerMapping, XGetPointerControl, XChangePointerControl.

NAME

XSetRegion — set the clip_mask of the graphics context to the specified region.

SYNOPSIS

```
XSetRegion(display, gc, r)
  Display *display;
  GC gc;
  Region r;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
r	Specifies the region.

DESCRIPTION

XSetRegion sets the *clip_mask* of the GC to the specified region. Thereafter, all output requests made with `gc` will be confined to the region.

Regions are located using an offset from an arbitrarily chosen point (the "region origin") which is common to all regions. It is up to the application to interpret the location of the region relative to a drawable. When the region is to be used as a *clip_mask* by calling *XSetRegion,* the top-left corner of region relative to the drawable used in the graphics request will be at *(xoffset + clip_x_origin, yoffset + clip_y_origin),* where *xoffset* and *yoffset* are the offset of the region and *clip_x_origin* and *clip_y_origin* are elements of the GC used in the graphics request.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XSetScreenSaver — set parameters of the screen saver.

SYNOPSIS

XSetScreenSaver (*display*, *timeout*, *interval*, *prefer_blanking*, *allow_exposures*)
  Display **display*;
  int *timeout*, *interval*;
  int *prefer_blanking*;
  int *allow_exposures*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
timeout	Specifies the time of inactivity, in seconds, before the screen saver turns on.
interval	Specifies the interval, in seconds, between screen saver invocations. This is for intermittent changes to the display, not blanking.
prefer_blanking	Specifies whether to enable screen blanking. Possible values are *DontPreferBlanking*, *PreferBlanking*, or *Default-Blanking*.
allow_exposures	Specifies the current screen saver control values. Possible values are *DontAllowExposures*, *AllowExposures*, or *Defaul-tExposures*.

DESCRIPTION

XSetScreenSaver sets the parameters that control the screen saver. *timeout* and *interval* are specified in seconds. A positive *timeout* enables the screen saver. A *timeout* of *0* disables the screen saver, while a timeout of *-1* restores the default. An *interval* of *0* disables the random pattern motion. If no input from devices (keyboard, mouse, etc.) is generated for the specified number of timeout seconds, the screen saver is activated.

For each screen, if blanking is preferred and the hardware supports video blanking, the screen will simply go blank. Otherwise, if either exposures are allowed or the screen can be regenerated without sending exposure events to clients, the screen is tiled with the root window background tile, with a random origin, each *interval* seconds. Otherwise, the state of the screen does not change. All screen states are restored at the next input from a device.

If the server-dependent screen saver method supports periodic change, *interval* serves as a hint about how long the change period should be, and zero hints that no periodic change should be made. Examples of ways to change the screen include scrambling the color map periodically, moving an icon image about the screen periodically, or tiling the screen with the root window background tile, randomly reoriginated periodically.

ERRORS

BadValue	*timeout* < –1.

SEE ALSO

XForceScreenSaver, XActivateScreenSaver, XResetScreenSaver, XGetScreenSaver.

NAME

XSetSelectionOwner — set owner of a selection.

SYNOPSIS

```
XSetSelectionOwner(display, selection, owner, time)
  Display *display;
  Atom selection;
  Window owner;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
selection	Specifies the selection atom. Predefined atoms are PRIMARY and SECONDARY.
owner	Specifies the present owner of the specified selection atom. This value is either a window ID or *None.*
time	Specifies the time when the grab should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime.*

DESCRIPTION

XSetSelectionOwner sets the *owner* and last-change time of a selection property. This should be called by an application that supports cutting and pasting between windows (or at least cutting), when the user has made a selection of any kind of text, graphics, or data. This makes the information available so that other applications can request the data from the new selection owner using *XConvertSelection,* which generates a *SelectionRequest* event specifying the desired type and format of the data. Then the selection owner sends a *SelectionNotify* using *XSendEvent* which notes that the information is stored in the selection property in the desired format or indicates that it couldn't do the conversion to the desired type.

If *owner* is specified as *None,* then the new owner of the selection is *None.* Otherwise, the new owner is the client executing the request.

If the new owner is not the same as the current owner of the selection, and the current owner is a window, then the current owner is sent a *SelectionClear* event. This indicates to that window that the selection should be unhighlighted.

If the selection owner window is later destroyed, the owner of the selection automatically reverts to *None.*

The value you pass to the *time* argument must be no earlier than the last-change time of the specified selection, and no later than the current time, or the selection is not affected. The new last-change time recorded is the specified time, with *CurrentTime* replaced by the current server time. If the X server reverts a selection owner to *None,* the last-change time is not affected.

ERRORS

BadAtom
BadWindow

SEE ALSO

XGetSelectionOwner, XConvertSelection.

NAME

XSetSizeHints — set the value of any property of type SIZE_HINTS.

SYNOPSIS

```
XSetSizeHints(display, w, hints, property)
  Display *display;
  Window w;
  XSizeHints *hints;
  Atom property;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
hints	Specifies a pointer to the size hints.
property	Specifies the property atom.

DESCRIPTION

XSetSizeHints sets the named property on the specified window with the *XSizeHints* structure. It is useful if new properties of type WM_SIZE_HINTS are defined. The pre-defined properties of that type have their own set and get functions, *XSetNormalHints* and *XSetZoomHints*.

STRUCTURES

```
typedef struct {
  long flags;     /* which fields in structure are defined */
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
          int x;              /* numerator */
          int y;              /* denominator */
  } min_aspect, max_aspect;
} XSizeHints;

/* flags argument in size hints */
#define USPosition (1L << 0)   /* user specified x, y */
#define USSize     (1L << 1)   /* user specified width, height */

#define PPosition  (1L << 2)   /* program specified position */
```

```
#define PSize       (1L << 3)   /* program specified size */
#define PMinSize    (1L << 4)   /* program specified minimum size */
#define PMaxSize    (1L << 5)   /* program specified maximum size */
#define PResizeInc  (1L << 6)   /* program specified resize increments */
#define PAspect     (1L << 7)   /* program specified min/max aspect ratios */
#define PAllHints  (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
```

ERRORS

BadAlloc
BadAtom
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XSetStandardColormap — create a standard colormap.

SYNOPSIS

```
void XSetStandardColormap(display, w, cmap, property)
  Display *display;
  Window w;
  XStandardColormap *cmap;
  Atom property;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
cmap	Specifies the filled colormap information structure.
property	Specifies the type of standard colormap desired. The predefined standard colormaps are: *XA_RGB_BEST_MAP, XA_RGB_RED_MAP, XA_RGB_GREEN_MAP, XA_RGB_BLUE_MAP, XA_DEFAULT_MAP,* and *XA_RGB_GRAY_MAP.*

DESCRIPTION

XSetStandardColormap is used to define a standard colormap if one is not already set as a property of the root window. It is usually used only by window managers. To create a standard colormap, follow this procedure:

1. Grab the server.
2. See if *property* is on the property list of the root window for the display. If it is, the colormap already exists.
3. If the desired property is not present, do the following:
   - Create a colormap (not required for RGB_DEFAULT_MAP)
   - Determine the color capabilities of the display.
   - Call *XAllocColorPlanes* or *XAllocColorCells* to allocate cells in the colormap.

- Call *XStoreColors* to store appropriate color values in the colormap.
- Fill in the descriptive fields in the structure.
- Call *XChangeProperty* to set the property on the root window.

4. Ungrab the server.

Refer to the *GSE Programmer's Guide* for a description of pre-defined standard colormap atoms.

ERRORS

BadAlloc
BadAtom
BadWindow

STRUCTURES

```
typedef struct {
  Colormap colormap;       /* ID of Colormap made by XCreateColormap *
  unsigned long red_max;
  unsigned long red_mult;
  unsigned long green_max;
  unsigned long green_mult;
  unsigned long blue_max;
  unsigned long blue_mult;
  unsigned long base_pixel;
} XStandardColormap;
```

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XFreeColormap, XGetStandardColormap, XInstallColormap, XUninstallColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XSetStandardProperties — set minimum set of properties for window manager.

SYNOPSIS

```
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
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
window_name	Specifies the name of the window.
icon_name	Specifies the name to be displayed in the window's icon.
icon_pixmap	Specifies the pixmap that is to be used for the icon, or *None*. This pixmap should normally be of depth one.
argv	Specifies a pointer to the command and arguments used to start the application.
argc	Specifies the number of arguments.
hints	Specifies a pointer to the sizing hints for the window in its normal state.

DESCRIPTION

XSetStandardProperties sets in a single call the most essential properties for a quickie application. *XSetStandardProperties* gives a window manager some information about your program's preferences; it probably will not be sufficient for complex programs. Refer to the *GSE Programmer's Guide* for a description of standard properties.

STRUCTURES

```
typedef struct {
  long flags;   /* which fields in structure are defined */
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
          int x;              /* numerator */
          int y;              /* denominator */
  } min_aspect, max_aspect;
} XSizeHints;

/* flags argument in size hints */
#define USPosition (1L << 0) /* user specified x, y */
#define USSize     (1L << 1) /* user specified width, height */

#define PPosition (1L << 2) /* program specified position */
#define PSize     (1L << 3) /* program specified size */

#define PMinSize (1L << 4) /* program specified minimum size */
#define PMaxSize (1L << 5) /* program specified maximum size */
#define PResizeInc (1L << 6) /* program specified resize increments */
#define PAspect (1L << 7) /* program specified min and max aspect ratios */
#define PAllHints (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspect)
```

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetFontProperty, XRotateWindowProperties, XDeleteProperty, XChangeProperty, XGetWindowProperty, XListProperties, XGetAtomName, XInternAtom.

NAME

XSetState — set foreground, background, logical function and plane mask in GC.

SYNOPSIS

```
XSetState(display, gc, foreground, background, function,
plane_mask)
  Display *display;
  GC gc;
  unsigned long foreground, background;
  int function;
  unsigned long plane_mask;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
foreground	Specifies the foreground you want for the specified graphics context.
background	Specifies the background you want for the specified graphics context.
function	Specifies the function you want for the specified graphics context.
plane_mask	Specifies the plane mask you want for the specified graphics context.

DESCRIPTION

XSetState sets the *foreground* and *background* pixel values, the logical *function,* and the *plane_mask* in a GC. See *XSetForeground, XSetBackground, XSetFunction,* and *XSetPlaneMask* for what these members do and for appropriate values.

Refer to the *GSE Programmer's Guide* for more information.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetSubwindowMode, DefaultGC.

NAME

XSetStipple — set stipple in graphics context.

SYNOPSIS

```
XSetStipple (display, gc, stipple)
  Display  *display;
  GC gc;
  Pixmap stipple;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
stipple	Specifies the stipple you want to set for the specified graphics context.

DESCRIPTION

XSetStipple sets the *stipple* member of the GC. The *stipple* is a pixmap of depth one. It is laid out like a tile. Set bits in the stipple determine which pixels in an area are drawn in the *foreground* pixel value. Unset bits in the stipple determine which pixels are drawn in the *background* pixel value if the *fill_style* is *FillOpaqueStippled*. If *fill_style* is *FillStippled*, pixels overlayed with unset bits in the stipple are not drawn. If *fill_style* is *FillTiled* or *FillSolid*, the stipple is not used.

ERRORS

BadAlloc
BadGC
BadMatch
BadPixmap

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetSubwindowMode — set subwindow mode in graphics context.

SYNOPSIS

```
XSetSubwindowMode(display, gc, subwindow_mode)
  Display *display;
  GC gc;
  int subwindow_mode;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
subwindow_mode	Specifies the subwindow mode you want to set for the specified graphics context. Possible values are *ClipByChildren* or *IncludeInferiors*.

DESCRIPTION

XSetSubwindowMode sets the *subwindow_mode* member of the GC. *ClipByChildren* means that graphics requests will be clipped by all viewable children. *IncludeInferiors* means draw through all subwindows.

ERRORS

BadGC
BadValue

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetTSOrigin, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, DefaultGC.

NAME

XSetTile — set fill tile in graphics context.

SYNOPSIS

```
XSetTile(display, gc, tile)
  Display *display;
  GC gc;
  Pixmap tile;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
tile	Specifies the desired tile for the specified graphics context.

DESCRIPTION

XSetTile sets the *tile* member of the GC. This member of the GC determines the pixmap used to tile areas. The tile must have the same depth as the destination drawable.

ERRORS

BadAlloc
BadGC
BadMatch
BadPixmap

SEE ALSO

XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XSetTransientForHint — set WM_TRANSIENT_FOR property for window.

SYNOPSIS

```
XSetTransientForHint(display, w, prop_window)
  Display *display;
  Window w;
  Window prop_window;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
prop_window	Specifies the window ID that the WM_TRANSIENT_FOR property is to be set to.

DESCRIPTION

XSetTransientForHint sets the WM_TRANSIENT_FOR property of the specified window. This should be done when the window *w* is a temporary child (for example, a dialog box) and the main top-level window of its application is *prop_window*. Some window managers may use this information to unmap an application's dialog boxes (for example, when the main application window gets iconified).

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XSetTSOrigin — set tile/stipple origin in graphics context.

SYNOPSIS

```
XSetTSOrigin(display, gc, ts_x_origin, ts_y_origin)
  Display *display;
  GC gc;
  int ts_x_origin, ts_y_origin;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
gc	Specifies the graphics context.
ts_x_origin *ts_y_origin*	Specify the x and y coordinates of the tile/stipple origin.

DESCRIPTION

XSetTSOrigin sets the *ts_x_origin* and *ts_y_origin* in the GC, which are measured relative to the origin of the drawable specified in the drawing request that uses the GC. This controls the placement of the tile or the stipple pattern that patterns an area. To tile or stipple a child so that the pattern matches the parent, you need to subtract the current position of the child window from *ts_x_origin* and *ts_y_origin*.

ERRORS

BadGC

SEE ALSO

XChangeGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XSetStipple, XSetPlaneMask, XSetDashes, XSetLineAttributes, XSetFillRule, XSetFillStyle, XSetForeground, XSetBackground, XSetFunction, XSetGraphicsExposures, XSetArcMode, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetState, XSetSubwindowMode, DefaultGC.

NAME

XSetWindowBackground — set the background pixel attribute of a window.

SYNOPSIS

```
XSetWindowBackground(display, w, background_pixel)
  Display *display;
  Window w;
  unsigned long background_pixel;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. Must be an *InputOutput* window.
background_pixel	Specifies which entry in the colormap is used as the background.

DESCRIPTION

XSetWindowBackground sets the *background* attribute of a window, setting the pixel value to be used to fill the background. The current window contents are not changed. The background is automatically repainted after *Expose* events, in the area affected by the exposure.

When *XSetWindowBackground* and *XSetWindowBackgroundPixmap* are both used on a window, whichever is called last will control the current background. It is an error to try to change the background of an *InputOnly* window.

ERRORS

BadMatch
BadWindow

SEE ALSO

XGetWindowAttributes, XChangeWindowAttributes, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap, XGetGeometry.

NAME

XSetWindowBackgroundPixmap — change background tile attribute of a window.

SYNOPSIS

```
XSetWindowBackgroundPixmap(display, w, background_tile)
  Display *display;
  Window w;
  Pixmap background_tile;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. Must be an *InputOutput* class window.
background_tile	Specifies a pixmap ID, *None*, or *ParentRelative* to be used as a background.

DESCRIPTION

XSetWindowBackgroundPixmap sets the *background_pixmap* attribute of a window. If no background pixmap is specified, the background pixmap of the window's parent is used. On the root window, the default background will be restored. The old, unused background pixmap can immediately be freed if no further explicit references to it are to be made.

XSetWindowBackgroundPixmap can only be performed on an *InputOutput* window. An error will result otherwise.

This does not change the current contents of the window, so you may wish to call *XClearWindow* to repaint the window after this function.

XSetWindowBackground may be used if a solid color instead of a tile is desired. If *background_tile* is specified as *ParentRelative,* the windows will get an *Expose* event when it is moved and its background will be repainted. When *XSetWindowBackground* and *XSetWindowBackgroundPixmap* are both used on a window, whichever is called last will control the current background.

ERRORS

BadColor
BadMatch
BadPixmap
BadWindow

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XSetWindowBorder — change window border attribute to pixel value and repaint border.

SYNOPSIS

```
XSetWindowBorder(display, w, border_pixel)
  Display *display;
  Window w;
  unsigned long border_pixel;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. Must be an *InputOutput* window.
border_pixel	Specifies the entry in the colormap. *XSetWindowBorder* uses this entry to paint the border.

DESCRIPTION

XSetWindowBorder sets the *border_pixel* attribute of window *w* to a pixel value, and repaints the border. The border is also automatically repainted after *Expose* events.

Use *XSetWindowBorderPixmap* to create a tiled border.

ERRORS

BadMatch
BadPixmap
BadValue
BadWindow

SEE ALSO

XGetWindowAttributes, XChangeWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorderPixmap, XGetGeometry.

NAME

XSetWindowBorderPixmap — change window border tile attribute and repaint border.

SYNOPSIS

```
XSetWindowBorderPixmap(display, w, border_tile)
  Display *display;
  Window w;
  Pixmap border_tile;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID of an *InputOutput* window.
border_tile	Specifies any pixmap or *None*.

DESCRIPTION

XSetWindowBorderPixmap sets the *border_pixmap* attribute of a window and repaints the border. The *border_tile* can be freed immediately after the call if no further explicit references to it are to be made.

This function can only be performed on an *InputOutput* window.

ERRORS

BadMatch
BadPixmap
BadValue
BadWindow

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XWriteBitmapFile, XReadBitmapFile, XCreateBitmapFromData.

NAME

XSetWindowBorderWidth — change the border width of a window.

SYNOPSIS

```
XSetWindowBorderWidth(display, w, width)
  Display *display;
  Window w;
  unsigned int width;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
width	Specifies the width of the window border.

DESCRIPTION

XSetWindowBorderWidth changes the border width of a window. This request is often used by the window manager as an indication of the current input focus window, so other clients should not change it.

SEE ALSO

XLowerWindow, XRaiseWindow, XCirculateSubwindows, XCirculateSubwindowsDown, XCirculateSubwindowsUp, XRestackWindows, XMoveWindow, XResizeWindow, XMoveResizeWindow, XReparentWindow, XConfigureWindow, XQueryTree.

NAME

XSetWindowColormap — set the colormap for a specified window.

SYNOPSIS

```
XSetWindowColormap(display, w, cmap)
  Display *display;
  Window w;
  Colormap cmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window to which you want to set the colormap.
cmap	Specifies the colormap.

DESCRIPTION

XSetWindowColormap sets the *colormap* attribute of the specified window. The colormap need not be installed to be set as an attribute. *cmap* will be used to translate pixel values drawn into this window if *cmap* is installed in the hardware.

Eventually, window managers will install and uninstall the proper colormaps according to this attribute and the pointer position or some other convention. For now, applications must install their own colormaps if they cannot use the default colormap.

ERRORS

BadColor
BadMatch
BadWindow

SEE ALSO

XGetWindowAttributes, XChangeWindowAttributes, XSetWindowBackground, XSetWindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap, XGetGeometry.

NAME

XSetWMHints — set the window manager hints property.

SYNOPSIS

```
XSetWMHints(display, w, wmhints)
  Display  *display;
  Window w;
  XWMHints  *wmhints;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
wmhints	Specifies a pointer to the window manager hints.

DESCRIPTION

XSetWMHints 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. Refer to the *GSE Programmer's Guide* for a description of each *XWMHints* structure member.

STRUCTURES

```
typedef struct {
  long flags; /* marks which fields in this structure are defined */
  Bool input; /* does application need window manager for keyboard input */
  int initial_state;  /* see below */
  Pixmap icon_pixmap; /* pixmap to be used as icon */
  Window icon_window; /* window to be used as icon */
  int icon_x, icon_y; /* initial position of icon */
  Pixmap icon_mask;   /* icon mask bitmap */
  XID window_group;   /* ID of related window group */
  /* this structure may be extended in the future */
} XWMHints;
```

```
/* definitions for the flags field: */
#define InputHint           (1L << 0)
#define StateHint           (1L << 1)
#define IconPixmapHint      (1L << 2)
#define IconWindowHint      (1L << 3)
#define IconPositionHint    (1L << 4)
#define IconMaskHint        (1L << 5)
#define WindowGroupHint     (1L << 6)
#define AllHints (InputHint|StateHint|IconPixmapHint|IconWindowHint| \'
  IconPositionHint|IconMaskHint|WindowGroupHint)

/* definitions for the initial state flag: */
#define DontCareState 0 /* don't know or care */
#define NormalState   1 /* most applications want to start this way */
#define ZoomState     2 /* application wants to start zoomed */
#define IconicState   3 /* application wants to start as an icon */
#define InactiveState 4 /* application believes it is seldom used;
                           some wm's may put it on inactive menu */
```

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XSetZoomHints — set size hints property for zoomed windows.

SYNOPSIS

XSetZoomHints(*display*, *w*, *zhints*)
  Display **display*;
  Window *w*;
  XSizeHints **zhints*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.
zhints	Specifies a pointer to the zoom hints.

DESCRIPTION

XSetZoomHints sets the WM_ZOOM_HINTS property for an application's top-level window in its zoomed state. Many window managers think of windows in three states: iconic, normal, or zoomed, corresponding to small, medium, and large.

To set size hints, an application must not only assign values to the appropriate elements in the hints structure, but also must set the *flags* field of the structure to indicate which members have assigned values and where it came from.

STRUCTURES

```
typedef struct {
  long flags;  /* which fields in structure are defined */
  int x, y;
  int width, height;
  int min_width, min_height;
  int max_width, max_height;
  int width_inc, height_inc;
  struct {
          int x;               /* numerator */
          int y;               /* denominator */
  } min_aspect, max_aspect;

/* flags argument in size hints */
#define USPosition (1L << 0) /* user specified x, y */
#define USSize     (1L << 1) /* user specified width, height */
```

```
#define PPosition  (1L << 2) /* program specified position */
#define PSize      (1L << 3) /* program specified size */
#define PMinSize   (1L << 4) /* program specified minimum size */
#define PMaxSize   (1L << 5) /* program specified maximum size */
#define PResizeInc (1L << 6) /* program specified resize increments */
#define PAspect    (1L << 7) /* program specified min/max aspect ratio
#define PAllHints (PPosition|PSize|PMinSize|PMaxSize|PResizeInc|PAspec
} XSizeHints;
```

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XStoreName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XShrinkRegion — reduce or expand the size of a region.

SYNOPSIS

```
XShrinkRegion(r, dx, dy)
  Region r;
  int dx, dy;
```

ARGUMENTS

r	Specifies the region.
dx	
dy	Specify the amounts by which you want to shrink or expand the specified region. Positive values expand the region while negative values shrink the region.

DESCRIPTION

XShrinkRegion changes the width of the specified region by the ratio *(currentwidth + dx)/currentwidth* and the height by the ratio *(currentheight + dy)/currentheight*. Counter to the name of the routine and the MIT documentation, the code seems to show that positive values expand the region; negative values shrink the region. The offset of the region is changed to keep the center of the resized region near its original position.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XStoreBuffer — store data in a cut buffer.

SYNOPSIS

```
XStoreBuffer(display, bytes, nbytes, buffer)
  Display *display;
  char bytes[];
  int nbytes;
  int buffer;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
bytes	Specifies the string of bytes you want stored. The byte string is not necessarily ASCII or null-terminated.
nbytes	Specifies the number of bytes in the string.
buffer	Specifies the cut buffer in which to store the byte string. Must be in the range 0 through 7.

DESCRIPTION

XStoreBuffer stores the specified data into one of the eight cut buffers. All eight buffers must be stored into before they can be circulated with *XRotateBuffers*. The cut buffers are numbered 0 through 7. Use *XFetchBuffer* to recover data from any cut buffer.

ERRORS

BadAlloc
BadAtom
BadWindow

SEE ALSO

XStoreBytes, XFetchBuffer, XFetchBytes, XRotateBuffers.

NAME

XStoreBytes — store data in cut buffer 0.

SYNOPSIS

```
XStoreBytes (display, bytes, nbytes)
  Display *display;
  char bytes[];
  int nbytes;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
bytes	Specifies the string of bytes you want stored. The byte string is not necessarily ASCII or null-terminated.
nbytes	Specifies the number of bytes that you want stored.

DESCRIPTION

XStoreBytes stores data in cut buffer 0, usually for reading by another client that already knows the meaning of the contents. Note that the cut buffer's contents need not be text, so null bytes are not special.

The cut buffer's contents may be retrieved later by any client calling *XFetchBytes*.

Use *XStoreBuffer* to store data in buffers 1 through 7.

ERRORS

BadAlloc
BadWindow

SEE ALSO

XStoreBuffer, XFetchBuffer, XFetchBytes, XRotateBuffers.

NAME

XStoreColor — set or change read/write entry of colormap to closest available hardware color.

SYNOPSIS

```
XStoreColor(display, cmap, colorcell_def)
  Display *display;
  Colormap cmap;
  XColor *colorcell_def;        /* send and RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
colorcell_def	Specifies a pixel value and the desired RGB values, and returns the RGB values actually used in the colormap, which will be the closest available on the hardware.

DESCRIPTION

XStoreColor changes the RGB values of a colormap entry specified by *colorcell_def.pixel* to the closest values available on the hardware, and returns these values to *colorcell_def*. This pixel value must be a read/write cell and a valid index into *cmap*. *XStoreColor* changes the red, green, and/or blue color components in the cell according to the *colorcell_def.flags* member, which you set by ORing the constants *DoRed*, *DoGreen*, and/or *DoBlue*.

If the colormap is an installed map for its screen, the changes are visible immediately.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                    /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadValue	*pixel* not a valid index into *cmap*.
BadColor	

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColors, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XStoreColors — change read/write colorcells to closest available hardware colors.

SYNOPSIS

```
XStoreColors(display, cmap, colorcell_defs, ncolors)
  Display *display;
  Colormap cmap;
  XColor colorcell_defs[ncolors];/* send and RETURN */
  int ncolors;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
colorcell_defs	Specifies an array of color definition structures.
ncolors	Specifies the number of *XColor* structures in *colorcell_defs*.

DESCRIPTION

XStoreColors changes the RGB values of each colormap entry specified by *colorcell_defs[].pixel* to the closest available hardware colors and returns these values to the RGB members of *colorcell_defs*. Each pixel value must be a read/write cell and a valid index into *cmap*. *XStoreColors* changes the red, green, and/or blue color components in each cell according to the *colorcell_defs[].flags* member, which you set by ORing the constants *DoRed, DoGreen,* and/or *DoBlue*. The specified pixels are changed if they are writable by any client, even if one or more pixels generates an error.

If the colormap is an installed map for its screen, the changes are visible immediately.

STRUCTURES

```
typedef struct {
  unsigned long pixel;
  unsigned short red, green, blue;
  char flags;                    /* DoRed, DoGreen, DoBlue */
  char pad;
} XColor;
```

ERRORS

BadAccess	A specified pixel is unallocated or read-only.
BadColor	
BadValue	A specified pixel is not a valid entry into *cmap*.

SEE ALSO

XAllocColorCells, XAllocColorPlanes, XAllocColor, XAllocNamedColor, XLookupColor, XParseColor, XQueryColor, XQueryColors, XStoreColor, XFreeColors, XStoreNamedColor, BlackPixel, WhitePixel.

NAME

XStoreName — assign name to window for window manager.

SYNOPSIS

```
XStoreName(display, w, window_name)
  Display *display;
  Window w;
  char *window_name;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window to which you want to assign a name.
window_name	Specifies the name of the window. The name should be a null-terminated string. This name is returned by any subsequent call to *XFetchName*.

DESCRIPTION

XStoreName sets the WM_NAME property, which should be used by the application to communicate the following information to the window manager, according to current conventions:

- To permit the user to identify one of a number of instances of the same client.
- To provide the user with non-critical state information.

Clients can assume that at least the beginning of this string is visible to the user. The WM_CLASS property, on the other hand, has two members which should be used to identify the client in general and each instance in particular. It is used for obtaining resources. See *XSetClassHint* for more details.

ERRORS

BadAlloc
BadWindow

SEE ALSO

XGetClassHint, XSetClassHint, XGetSizeHints, XSetSizeHints, XGetWMHints, XSetWMHints, XGetZoomHints, XSetZoomHints, XGetNormalHints, XSetNormalHints, XGetTransientForHint, XSetTransientForHint, XFetchName, XGetIconName, XSetIconName, XGetIconSizes, XSetIconSizes, XSetCommand.

NAME

XStoreNamedColor — allocate read/write colorcell by color name.

SYNOPSIS

```
XStoreNamedColor(display, cmap, color, pixel, flags)
  Display *display;
  Colormap cmap;
  char *color;
  unsigned long pixel;
  int flags;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.
color	Specifies the color name string (for example, "red"). ISO Latin-1 encoding, upper/lower case is not important.
pixel	Specifies the entry in the colormap to store color in.
flags	Specifies which red, green, and blue indexes are set.

DESCRIPTION

XStoreNamedColor looks up the named *color* in the database, with respect to the screen associated with *cmap*, then stores the result in the cell of *cmap* specified by *pixel*. Upper/lowercase in name does not matter. The *flags* argument, a bitwise OR of the constants *DoRed*, *DoGreen*, and *DoBlue*, determines which subfields within the pixel value in the cell are written. The database is **/usr/lib/rgb**.

ERRORS

BadAccess	*pixel* is unallocated or read-only.
BadColor	
BadName	
BadValue	*pixel* is not a valid index into *cmap*.

Note: if more than one pixel is in error, the one reported is arbitrary.

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XFreeColormap, XGetStandardColormap, XInstallColormap, XUninstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XStringToKeysym — convert keysym name to keysym code.

SYNOPSIS

```
KeySym  XStringToKeysym(string)
  char  *string;
```

ARGUMENTS

string	Specifies the name of the keysym that is to be converted.

DESCRIPTION

XStringToKeysym translates the character string version of a keysym name to the matching *KeySym* (a number). Valid keysym names are listed in *<X11/keysymdef.h>*. If the specified string does not match a valid keysym, *XStringToKeysym* returns *NoSymbol.*

SEE ALSO

XDeleteModifiermapEntry, XInsertModifiermapEntry, XFreeModifiermap, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XNewModifierMap, XQueryKeymap, XLookupKeysym, XRebindKeysym, XGetKeyboardMapping, XChangeKeyboardMapping, XRefreshKeyboardMapping, XLookupString, XSetModifierMapping, XGetModifierMapping.

NAME

XSubImage — create subimage from part of image.

SYNOPSIS

```
XImage  *XSubImage(ximage, x, y, subimage_width,
subimage_height)
  XImage  *ximage;
  int x;
  int y;
  int subimage_width;
  int subimage_height;
```

ARGUMENTS

ximage	Specifies a pointer to the image.
x *y*	Specify the x and y coordinates of the origin of the subimage.
subimage_width *subimage_height*	Specify the width and height (in pixels) of the new subimage.

DESCRIPTION

XSubImage creates a new image that is a subsection of an existing one. It allocates the memory necessary for the new *XImage* structure and returns a pointer to the new image. The algorithm used is repetitive calls to *XGetPixel* and *XPutPixel*. Therefore, this function may be very slow.

XSubImage extracts a subimage from an image, while *XGetSubImage* extracts an image from a drawable.

SEE ALSO

XDestroyImage, XPutImage, XGetImage, XCreateImage, XGetSubImage, XAddPixel, XPutPixel, XGetPixel, ImageByteOrder.

NAME

XSubtractRegion — subtract one region from another.

SYNOPSIS

```
XSubtractRegion(sra, srb, dr)
  Region sra, srb;
  Region  dr;                /*  RETURN  */
```

ARGUMENTS

sra	
srb	Specify the two regions in which you want to perform the computation.
dr	Returns the result of the computation.

DESCRIPTION

XSubtractRegion calculates the difference between the two regions specified (*sra* – *srb*) and puts the result in *dr*.

This function returns a region which contains all parts of *sra* that are not also in *srb*.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRegion, XUnionRectWithRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XSync — flush output buffer and wait for all events and errors to be processed by server.

SYNOPSIS

```
XSync(display, discard)
  Display *display;
  int discard;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
discard	Specifies whether *XSync* discards all events on the input queue. This argument is either *True* or *False*.

DESCRIPTION

XSync flushes the output buffer, then waits until all events and errors resulting from previous calls have been received and processed by the X server. Events (and errors) are placed on the input queue. The client's *XError* subroutine is called once for EACH error received.

If discard is *True, XSync* discards all events on the input queue (including those events that were on the queue before *XSync* was called).

XSync is sometimes used with window manipulation functions (by the window manager) to wait for all resulting exposure events. Very few clients need to use this subroutine.

SEE ALSO

XFlush

NAME

XSynchronize — enable or disable synchronization for debugging.

SYNOPSIS

int (*XSynchronize(*display, onoff***))()**
  Display **display*;
  int *onoff*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
onoff	Specifies whether to enable or disable synchronization. You can pass 0 (disable synchronization) or non-zero (enable synchronization).

DESCRIPTION

XSynchronize turns on or off synchronous mode for debugging. If *onoff* is non-zero, it turns on synchronous behavior; *0* resets the state to off.

When events are synchronized, they are reported as they occur instead of at some later time, but server performance is many times slower. This can be useful for debugging complex event handling routines. Under SYSTEM V/68 the same result can be achieved without hardcoding by setting the global variable *_Xdebug* to *True*.

If synchronous mode was off before the call, *XSynchronize* returns NULL.

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XWindowEvent, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSendEvent, QLength.

NAME

XTextExtents — get string and font metrics.

SYNOPSIS

```
XTextExtents (font_struct, string, nchars, direction, ascent, descent,
overall)
  XFontStruct  *font_struct;
  char  *string;
  int nchars;
  int  *direction;              /*  RETURN  */
  int  *ascent,  *descent;      /*  RETURN  */
  XCharStruct  *overall;        /*  RETURN  */
```

ARGUMENTS

font_struct	Specifies a pointer to the *XFontStruct* structure.
string	Specifies the character string.
nchars	Specifies the number of characters in the character string.
direction	Returns the value of the direction element of the *XFontStruct*. Either *FontRightToLeft* or *FontLeftToRight.*
ascent	Returns the font ascent element of the *XFontStruct*. This is the overall maximum ascent for the font.
descent	Returns the font descent element of the *XFontStruct*. This is the overall maximum descent for the font.
overall	Returns the overall characteristics of the string. These are the sum of the *width* measurements for each character, the maximum *ascent* and *descent,* the minimum *lbearing* added to the width of all characters up to the character with the smallest *lbearing,* and the maximum *rbearing* added to the width of all characters up to the character with the largest *rbearing.*

DESCRIPTION

XTextExtents returns the dimensions in pixels that specify the bounding box of the specified string of characters in the named font, and the maximum ascent and descent for the entire font. This function performs the size computation locally and, thereby, avoids the roundtrip overhead of *XQueryTextExtents,* but it requires a filled *XFontStruct.*

ascent and *descent* return information about the font, while *overall* returns information about the given string. The returned *ascent* and *descent* should usually be used to calculate the line spacing, while the *width, rbearing,* and *lbearing* members of *overall* should be used for horizontal measures. The total height of the bounding rectangle, good for any string in this font, is *ascent* + *descent*.

overall.ascent is the maximum of the ascent metrics of all characters in the string. The *overall.descent* is the maximum of the descent metrics. The *overall.width* is the sum of the character-width metrics of all characters in the string. The *overall.lbearing* is the *lbearing* of the character in the string with the smallest *lbearing* plus the width of all the characters up to but not including that character. The *overall.rbearing* is the *rbearing* of the character in the string with the largest *rbearing* plus the width of all the characters up to but not including that character.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;      /* hook for extension to hang data */
  Font fid;                /* Font ID for this font */
  unsigned direction;      /* hint about direction the font is painted
  unsigned min_char_or_byte2; /* first character */
  unsigned max_char_or_byte2; /* last character */
  unsigned min_byte1;      /* first row that exists */
  unsigned max_byte1;      /* last row that exists */
  Bool all_chars_exist;    /* flag if all characters have non-zero siz
  unsigned default_char;   /* char to print for undefined character */
  int n_properties;        /* how many properties there are */
  XFontProp *properties;   /* pointer to array of additional propertie
  XCharStruct min_bounds; /* minimum bounds over all existing char*/
  XCharStruct max_bounds; /* minimum bounds over all existing char*/
  XCharStruct *per_char;  /* first_char to last_char information */
  int ascent;              /* logical extent above baseline for spacii
  int descent;             /* logical descent below baseline for spac:
} XFontStruct;

typedef struct {
  short lbearing;           /* origin to left edge of raster */
  short rbearing;           /* origin to right edge of raster */
  short width;              /* advance to next char's origin */
  short ascent;             /* baseline to top edge of raster */
  short descent;            /* baseline to bottom edge of raster */
  unsigned short attributes;/* per char flags (not predefined) */
} XCharStruct;
```

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents16, XTextWidth, XTextWidth16.

NAME

XTextExtents16 — get string and font metrics of 16-bit character string.

SYNOPSIS

```
XTextExtents16(font_struct, string, nchars, direction, ascent,
descent, overall)
  XFontStruct *font_struct;
  XChar2b *string;
  int nchars;
  int *direction;              /* RETURN */
  int *ascent, *descent;       /* RETURN */
  XCharStruct *overall;        /* RETURN */
```

ARGUMENTS

font_struct	Specifies a pointer to the *XFontStruct* structure.
string	Specifies the character string made up of *XChar26* structures.
nchars	Specifies the number of characters in the character string.
direction	Returns the value of the direction element of the *XFontStruct*. *FontRightToLeft* of *FontLeftToRight.*
ascent	Returns the font ascent element of the *XFontStruct*. This is the overall maximum ascent for the font.
descent	Returns the font descent element of the *XFontStruct*. This is the overall maximum descent for the font.
overall	Returns the overall characteristics of the string. These are the sum of the *width* measurements for each character, the maximum *ascent* and *descent,* the minimum *lbearing* added to the width of all characters up to the character with the smallest *lbearing,* and the maximum *rbearing* added to the width of all characters up to the character with the largest *rbearing.*

DESCRIPTION

XTextExtents16 returns the dimensions in pixels that specify the bounding box of the specified string of characters in the named font, and the maximum ascent and descent for the entire font. This function performs the size computation locally and, thereby, avoids the roundtrip overhead of *XQueryTextExtents16,* but it requires a filled *XFontStruct.*

ascent and *descent* return information about the font, while *overall* returns information about the given string. The returned *ascent* and *descent* should usually be used to calculate the line spacing, while the *width, rbearing,* and *lbearing* members of *overall* should be used for horizontal measures. The total height of the bounding rectangle, good for any string in this font, is *ascent + descent.*

overall.ascent is the maximum of the ascent metrics of all characters in the string. The *overall.descent* is the maximum of the descent metrics. The *overall.width* is the sum of the character-width metrics of all characters in the string. The *overall.lbearing* is the *lbearing* of the character in the string with the smallest *lbearing* plus the width of all the characters up to but not including that character. The *overall.rbearing* is the *rbearing* of the character in the string with the largest *rbearing* plus the width of all the characters up to but not including that character.

STRUCTURES

```
typedef struct {
  short lbearing; /* origin to left edge of raster*/
  short rbearing; /* origin to right edge of raster*/
  short width;    /* advance to next char's origin*/
  short ascent;   /* baseline to top edge of raster*/
  short descent;  /* baseline to bottom edge of raster*/
  unsigned short attributes; /* per char flags (not predefined)*/
} XCharStruct;

typedef struct {
  XExtData *ext_data;     /* hook for extension to hang data*/
  Font fid;               /* Font ID for this font*/
  unsigned direction;     /* hint about direction the font is painted*/
  unsigned min_char_or_byte2; /* first character*/
  unsigned max_char_or_byte2; /* last character*/
  unsigned min_byte1;     /* first row that exists*/
  unsigned max_byte1;     /* last row that exists*/
  Bool all_chars_exist;   /* flag if all characters have non-zero size*/
  unsigned default_char; /* char to print for undefined character*/
  int n_properties;       /* how many properties there are*/
  XFontProp *properties; /* pointer to array of additional properies*/
  XCharStruct min_bounds;/* minimum bounds over all existing char*/
  XCharStruct max_bounds;/* minimum bounds over all existing char*/
  XCharStruct *per_char; /* first_char to last_char information*/
  int ascent;             /* logical extent above baseline for spacing*/
```

```
    int descent;  /* logical descent below baseline for spacing*/
} XFontStruct;

typedef struct {       /* normal 16 bit characters are two bytes*/
     unsigned char byte1;
     unsigned char byte2;
} XChar2b;
```

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextWidth, XTextWidth16.

NAME

XTextWidth — get width in pixels of 8-bit character string.

SYNOPSIS

```
int XTextWidth(font_struct, string, count)
  XFontStruct *font_struct;
  char *string;
  int count;
```

ARGUMENTS

font_struct	Specifies the font description structure of the font in which you want to draw the string.
string	Specifies the character string.
count	Specifies the character count in *string*.

DESCRIPTION

XTextWidth returns the width in pixels of the specified string using the specified font. This is the sum of the *XCharStruct.width* for each character in the string. This is also equivalent to the value of *overall.width* returned by *XQueryTextExtents* or *XTextExtents.* The characters in *string* are 8-bit characters.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;     /* hook for extension to hang data*/
  Font fid;               /* Font ID for this font*/
  unsigned direction;     /* hint about direction the font is painted*/
  unsigned min_char_or_byte2;/* first character*/
  unsigned max_char_or_byte2;/* last character*/
  unsigned min_byte1;     /* first row that exists*/
  unsigned max_byte1;     /* last row that exists*/
  Bool all_chars_exist;   /* flag if all characters have non-zero size*/
  unsigned default_char;  /* char to print for undefined character*/
  int n_properties;       /* how many properties there are*/
  XFontProp *properties;  /* pointer to array of additional properties*/
  XCharStruct min_bounds; /* minimum bounds over all existing char*/
  XCharStruct max_bounds; /* minimum bounds over all existing char*/
  XCharStruct *per_char;  /* first_char to last_char information*/
  int ascent;             /* logical extent above baseline for spacing*/
  int descent;            /* logical descent below baseline for spacing*/
} XFontStruct;
```

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth16.

NAME

XTextWidth16 — get width in pixels of 16-bit character string.

SYNOPSIS

```
int XTextWidth16(font_struct, string, count)
  XFontStruct *font_struct;
  XChar2b *string;
  int count;
```

ARGUMENTS

font_struct	Specifies the font description structure of the font in which you want to draw the string.
string	Specifies the character string made up of *XChar2b* structures.
count	Specifies the character count in *string*.

DESCRIPTION

XTextWidth16 returns the width in pixels of the specified string using the specified font. This is the sum of the *XCharStruct.width* for each character in the string. This is also equivalent to the value of *overall.width* returned by *XQueryTextExtents16* or *XTextExtents16*.

The characters in *string* are 16-bit characters.

STRUCTURES

```
typedef struct {
  XExtData *ext_data;       /* hook for extension to hang data*/
  Font fid;                 /* Font ID for this font*/
  unsigned direction;       /* hint about direction the font is painted*/
  unsigned min_char_or_byte2;/* first character*/
  unsigned max_char_or_byte2;/* last character*/
  unsigned min_byte1;       /* first row that exists*/
  unsigned max_byte1;       /* last row that exists*/
  Bool all_chars_exist;     /* flag if all characters have non-zero size*/
  unsigned default_char;    /* char to print for undefined character*/
  int n_properties;         /* how many properties there are*/
  XFontProp *properties;    /* pointer to array of additional properties*/
  XCharStruct min_bounds;   /* minimum bounds over all existing char*/
  XCharStruct max_bounds;   /* minimum bounds over all existing char*/
  XCharStruct *per_char;    /* first_char to last_char information*/
  int ascent;               /* logical extent above baseline for spacing*/
  int descent;              /* logical descent below baseline for spacing*/
} XFontStruct;
```

SEE ALSO

XQueryTextExtents, XQueryTextExtents16, XDrawImageString, XDrawImageString16, XDrawString, XDrawString16, XDrawText, XDrawText16, XTextExtents, XTextExtents16, XTextWidth,

NAME

XTranslateCoordinates — change coordinate system from one window to another.

SYNOPSIS

```
int XTranslateCoordinates(display, src_w, dest_w, src_x,
src_y, dest_x, dest_y, child)
  Display *display;
  Window src_w, dest_w;
  int src_x, src_y;
  int *dest_x, *dest_y;        /* RETURN */
  Window *child;               /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
src_w	Specifies the window ID of the source window.
dest_w	Specifies the window ID of the destination window.
src_x *src_y*	Specify the x and y coordinates within the source window.
dest_x *dest_y*	Return the translated x and y coordinates within the destination window.
child	If the point is contained in a mapped child of the destination window, then that child ID is returned in *child*.

DESCRIPTION

XTranslateCoordinates translates coordinates from the frame of reference of one window to another. This should be avoided in most applications since it is a roundtrip request to the server. Most applications benefit from the window-based coordinate system anyway and don't need global coordinates.

XTranslateCoordinates returns *0*, if *src_w* and *dest_w* are on different screens, and **dest_x* and **dest_y* are *0*. In addition, if the coordinates are contained in a mapped child of *dest_w*, then that child is returned in the *child* argument. Otherwise, *XTranslateCoordinates* returns a non-zero value and sets **dest_x* and **dest_y* to the location of the point relative to *dest_w*.

Window managers often need to perform a coordinate transformation from the coordinate space of one window to another, or unambiguously determine which subwindow a coordinate lies in. *XTranslateCoordinates* fulfills this need, while avoiding any race conditions by asking the server to perform this operation.

ERRORS

BadWindow

SEE ALSO

XGeometry, XParseGeometry.

NAME

XUndefineCursor — disassociate cursor from window.

SYNOPSIS

```
XUndefineCursor(display, w)
  Display  *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XUndefineCursor sets the cursor for a window to its parent's cursor, undoing the effect of a previous *XDefineCursor* for this window. On the root window, with no cursor specified, the default cursor is restored.

ERRORS

BadWindow

SEE ALSO

XDefineCursor, XCreateFontCursor, XCreateGlyphCursor, XCreatePixmapCursor, XFreeCursor, XRecolorCursor, XQueryBestCursor, XQueryBestSize.

NAME

XUngrabButton — release a button from grab.

SYNOPSIS

```
XUngrabButton(display, button, modifiers, w)
  Display *display;
  unsigned int button;
  unsigned int modifiers;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
button	Specifies the mouse button. Specify *Button1, Button2, Button3, Button4, Button5,* or the constant *AnyButton,* which is equivalent to issuing the ungrab request for all possible buttons.
modifiers	Specifies a set of keymasks. This is a bitwise OR of one or more of the following symbols: *ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask,* or *AnyModifier. AnyModifier* is equivalent to issuing the ungrab button request for all possible modifier combinations (including no modifiers).
w	Specifies the window ID of the window you want to release the button grab.

DESCRIPTION

XUngrabButton cancels the passive grab on a button/key combination on the specified window if it was grabbed by this client. A *modifiers* of *AnyModifier* is equivalent to issuing the ungrab request for all possible modifier combinations (including the combination of no modifiers). A *button* of *AnyButton* is equivalent to issuing the request for all possible buttons. This call has no effect on an active grab.

ERRORS

BadWindow

SEE ALSO

XGrabKey, XUngrabKey, XGrabKeyboard, XUngrabKeyboard, XGrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XUngrabKey — release a key from grab.

SYNOPSIS

```
XUngrabKey(display, keycode, modifiers, w)
  Display *display;
  int keycode;
  unsigned int modifiers;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
keycode	Specifies the keycode. This keycode maps to the specific key you want to ungrab. Pass either a keycode or *AnyKey*.
modifiers	Specifies a set of keymasks. This is a bitwise OR of one or more of the following symbols: *ShiftMask, LockMask, ControlMask, Mod1Mask, Mod2Mask, Mod3Mask, Mod4Mask, Mod5Mask,* or *AnyModifier*. *AnyModifier* is equivalent to issuing the ungrab key request for all possible modifier combinations (including no modifiers).
w	Specifies the window ID of the window which you want to ungrab the specified keys.

DESCRIPTION

XUngrabKey cancels the passive grab on the key combination on the specified window if it was grabbed by this client. A *modifiers* of *AnyModifier* is equivalent to issuing the request for all possible modifier combinations (including the combination of no modifiers). A *keycode* of *AnyKey* is equivalent to issuing the request for all possible non-modifier key codes. This call has no effect on an active grab.

ERRORS

BadWindow

SEE ALSO

XGrabKey, XGrabKeyboard, XUngrabKeyboard, XGrabButton, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XUngrabKeyboard — release keyboard from grab.

SYNOPSIS

```
XUngrabKeyboard(display, time)
  Display *display;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
time	Specifies the time. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime*. If this time is earlier than the last-keyboard-grab time or later than the current server time, the keyboard will not be ungrabbed.

DESCRIPTION

XUngrabKeyboard releases any active grab on the keyboard by this client. It executes as follows:

- Releases the keyboard and any queued events if this client has it actively grabbed from either *XGrabKeyboard* or *XGrabKey*.
- Does not release the keyboard and any queued events if *time* is earlier than the last-keyboard-grab time or is later than the current X server time.
- Generates *FocusIn* and *FocusOut* events.

The X server automatically performs an *UngrabKeyboard* if the *event_window* that initiated an active keyboard grab becomes not viewable.

SEE ALSO

XGrabKey, XUngrabKey, XGrabKeyboard, XGrabButton, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer, XUngrabServer.

NAME

XUngrabPointer — release the pointer from grab.

SYNOPSIS

```
XUngrabKeyboard(display, time)
  Display *display;
  Time time;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
time	Specifies the time when the grab should take place. Pass either a timestamp, expressed in milliseconds, or the constant *CurrentTime*. If this time is earlier than the last-pointer-grab time or later than current server time, the pointer will not be grabbed.

DESCRIPTION

XUngrabPointer releases an active grab on the pointer by the calling client. It executes as follows:

- Releases the pointer and any queued events, if this client has actively grabbed the pointer from *XGrabPointer, XGrabButton,* or from a normal button press.
- Does not release the pointer if the specified time is earlier than the last-pointer-grab time or is later than the current X server time.
- Generates *EnterNotify* and *LeaveNotify* events.

The X server performs an *XUngrabPointer* automatically if the *event_window* or *confine_to* window for an active pointer grab becomes not viewable.

SEE ALSO

XQueryPointer, XWarpPointer, XGrabPointer, XChangeActivePointerGrab, XGetPointerMapping, XSetPointerMapping, XGetPointerControl, XChangePointerControl.

NAME

XUngrabServer — release server from grab.

SYNOPSIS

```
XUngrabServer(display)
  Display  *display;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.

DESCRIPTION

XUngrabServer releases the grabbed server, and begins execution of all the graphics requests queued during the grab. *XUngrabServer* is called automatically when a client closes its connection.

SEE ALSO

XGrabKey, XUngrabKey, XGrabKeyboard, XUngrabKeyboard, XGrabButton, XUngrabButton, XGrabPointer, XUngrabPointer, XChangeActivePointerGrab, XGrabServer.

NAME

XUninstallColormap — uninstall colormap, install default if not already installed.

SYNOPSIS

```
XUninstallColormap(display, cmap)
  Display *display;
  Colormap cmap;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
cmap	Specifies the colormap.

DESCRIPTION

If *cmap* is an installed map for its screen, it is uninstalled. If the screen's default colormap is not installed, it is installed.

If *cmap* is an installed map, a *ColormapNotify* event is generated on every window having this colormap as an attribute. If a colormap is installed as a result of the uninstall, a *ColormapNotify* event is generated on every window having that colormap as an attribute.

At any time, there is a subset of the installed colormaps, viewed as an ordered list (the "required list"). The length of the required list is at most the *min_maps* specified for each screen in the *Display* structure. When a colormap is installed with *XInstallColormap,* it is added to the head of the required list and the last colormap in the list is removed if necessary to keep the length of the list at *mim_maps.* When a colormap is uninstalled with *XUninstallColormap* and it is in the required list, it is removed from the list. No other actions by the server or the client change the required list. It is important to realize that on all but high-end workstations, *min_maps* is likely to be 1.

SEE ALSO

XCopyColormapAndFree, XCreateColormap, XFreeColormap, XGetStandardColormap, XInstallColormap, XSetStandardColormap, XListInstalledColormaps, XSetWindowColormap, DefaultColormap, DisplayCells.

NAME

XUnionRectWithRegion — add rectangle to region.

SYNOPSIS

```
XUnionRectWithRegion(rectangle, src_region, dest_region)
        XRectangle *rectangle;
        Region src_region;
        Region dest_region;
```

ARGUMENTS

rectangle	Specifies the rectangle to add to the region.
src_region	Specifies the source region to be used.
dest_region	Specifies the resulting region. May be the same as *src_region.*

DESCRIPTION

XUnionRectWithRegion computes the destination region from a union of the specified rectangle and the specified source region. The source and destination regions may be the same.

One common application of this function is to simplify the combining of the rectangles specified in *Expose* events into a *clip_mask* in the GC, thus restricting the redrawn areas to the exposed rectangles. Use *XUnion-RectWithRegion* to combine the rectangle in each *Expose* event into a region, then call *XSetRegion. XSetRegion* sets the *clip_mask* in a GC to the region. In this case, *src_region* and *dest_region* would be the same region.

If *src_region* and *dest_region* are not the same region, *src_region* is copied to *dest_region* before the rectangle is added to *dest_region.*

For more information on regions, refer to the *GSE Programmer's Guide.*

STRUCTURES

```
typedef struct {
    short x, y;
    unsigned short width, height;
} XRectangle;

The Region type is a pointer to an opaque data type.
Its definition is not needed by programs.
```

SEE ALSO

XXorRegion, XUnionRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XUnionRegion — compute the union of two regions.

SYNOPSIS

```
XUnionRegion(sra, srb, dr)
  Region sra, srb;
  Region  dr;
```

ARGUMENTS

sra	
srb	Specify the two regions in which you want to perform the computation.
dr	Returns the result of the computation.

DESCRIPTION

XUnionRegion computes the union of two regions and places the result in *dr*. The resulting region will contain all the area of both the source regions.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XXorRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

XUniqueContext — create a new context ID (not graphics context).

SYNOPSIS

```
XContext  XUniqueContext()
```

DESCRIPTION

The context manager allows association of arbitrary data with a resource ID. This call creates an instance of the *XContext* structure with a unique resource ID that will be used in subsequent calls to *XFindContext, XDeleteContext* and *XSaveContext.*

STRUCTURES

```
typedef int XContext;
```

SEE ALSO

XDeleteContext, XFindContext, XSaveContext.

NAME

XUnloadFont — unload a font.

SYNOPSIS

```
XUnloadFont(display, font)
  Display *display;
  Font font;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
font	Specifies the font ID.

DESCRIPTION

XUnloadFont indicates to the server that this client no longer needs the specified font. The font may be unloaded on the X server if this is the last client that needs the font. In any case, the font should never again be referenced by this client because X destroys the resource ID.

ERRORS

BadFont

SEE ALSO

XLoadFont, XLoadQueryFont, XFreeFont, XFreeFontInfo, XListFonts, XListFontsWithInfo, XFreeFontNames, XFreeFontPath, XGetFontPath, XQueryFont, XSetFont, XSetFontPath, XGetFontProperty, XCreateFontCursor.

NAME

XUnmapSubwindows — unmap all subwindows of a given window.

SYNOPSIS

```
XUnmapSubwindows(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XUnmapSubwindows performs an *XUnmapWindow* on all mapped children of *w*, in bottom to top stacking order.

XUnmapSubwindows also generates an *UnmapNotify* event on each subwindow and generates exposure events on formerly obscured windows. This is much more efficient than unmapping many subwindows one at a time, as much of the work need only be performed once for all of the subwindows rather than for each subwindow.

ERRORS

BadWindow

SEE ALSO

XMapRaised, XMapSubwindows, XMapWindow, XUnmapWindow.

NAME

XUnmapWindow — unmaps a window.

SYNOPSIS

```
XUnmapWindow(display, w)
  Display *display;
  Window w;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID.

DESCRIPTION

XUnmapWindow removes *w* and all its descendants from the screen. If *w* is already unmapped, *XUnmapWindow* has no effect. Otherwise, *w* is unmapped and an *UnmapNotify* event is generated. Normal exposure processing on formerly obscured windows is performed.

Descendants of *w* will not be visible until *w* is mapped again. In other words, the subwindows are still mapped, but are not visible because *w* is unmapped. Unmapping a *w* will generate exposure events on windows that were formerly obscured by *w* and its children.

ERRORS

BadWindow

SEE ALSO

XMapRaised, XMapSubwindows, XMapWindow, XUnmapSubwindows.

NAME

XWarpPointer — move the pointer to another point on screen.

SYNOPSIS

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*, *dest_x*;
  unsigned int *src_width*, *src_height*;
  int *dest_x*, *dest_y*;

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
src_w	Specifies the window ID of the source window. You can also pass *None*.
dest_w	Specifies the window ID of the destination window. You can also pass *None*.
src_x *src_y*	Specify the x and y coordinates within the source window. These are used with *src_width* and *src_height* to determine the rectangle the pointer must be in. They are not the present pointer position. If *src_y* is *None*, these coordinates are relative to the root window of *src_w*.
src_width *src_height*	Specify the width and height of the source window. Used with *src_x* and *src_y*.
dest_x *dest_y*	Specify the destination x and y coordinates within the destination window. If *dest_y* is *None*, these coordinates are relative to the root window of *dest_w*.

DESCRIPTION

XWarpPointer moves the pointer suddenly from one point on the screen to another.

If *dest_w* is a window, *XWarpPointer* moves the pointer to [*dest_x, dest_y*] relative to the destination window's origin. If *dest_w* is *None*, *XWarpPointer* moves the pointer according to the offsets [*dest_x, dest_y*] relative to the current position of the pointer.

If *src_window* is *None,* the move is independent of the current cursor position (*dest_x* and *dest_y* use global coordinates). If the source window is not *None,* the move only takes place if the pointer is currently contained in a visible portion of the rectangle of the source window (including its inferiors) specified by *src_x, src_y, src_width* and *src_height.* If *src_width* is zero (0), the pointer must be between *src_x* and the right edge of the window to be moved. If *src_height* is zero (0), the pointer must be between *src_y* and the bottom edge of the window to be moved.

XWarpPointer cannot be used to move the pointer outside the *confine_to* window of an active pointer grab. If this is attempted the pointer will be moved to the point on the border of the *confine_to* window nearest the requested destination.

XWarpPointer generates events as if the user had (instantaneously) moved the pointer.

This function should not be used unless absolutely necessary, and then only in tightly controlled, predictable situations. It has the potential to confuse the user.

ERRORS

BadWindow

SEE ALSO

XQueryPointer, XGrabPointer, XChangeActivePointerGrab, XUngrabPointer, XGetPointerMapping, XSetPointerMapping, XGetPointerControl, XChangePointerControl.

NAME

XWindowEvent — remove next event matching mask and window.

SYNOPSIS

```
XWindowEvent(display, w, event_mask, rep)
  Display *display;
  Window w;
  long event_mask;
  XEvent *rep;              /* RETURN */
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay*.
w	Specifies the window ID. This is the window whose next matched event you want to remove.
event_mask	Specifies the event mask. See *XSelectInput* for a complete list of event masks.
rep	Specifies the event removed from the input queue. *XWindowEvent* returns this event to this argument.

DESCRIPTION

XWindowEvent searches the event queue for specific event types from the specified window. *XWindowEvent* removes the next event in the queue which matches both the passed window and the passed mask. The event is copied into an *XEvent* supplied by the caller. Other events in the queue are not discarded. If no such event has been queued, *XWindowEvent* flushes the output buffer and waits until one is received.

For Release 2, the output buffer is flushed only if no matching events are found on the queue. This change is compatible with applications written for Release 1.

STRUCTURES

See individual event structures described in the *GSE Programmer's Guide.*

SEE ALSO

XSelectInput, XSetInputFocus, XGetInputFocus, XCheckWindowEvent, XCheckTypedEvent, XCheckTypedWindowEvent, XMaskEvent, XCheckMaskEvent, XNextEvent, XEventsQueued, XAllowEvents, XGetMotionEvents, XIfEvent, XCheckIfEvent, XPeekEvent, XPeekIfEvent, XPutBackEvent, XPending, XSynchronize, XSendEvent, QLength.

NAME

XWriteBitmapFile — write bitmap to file.

SYNOPSIS

```
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;
```

ARGUMENTS

display	Specifies a pointer to the *Display* structure; returned from *XOpenDisplay.*
filename	Specifies the file name to use. The format of the file name is operating system specific.
bitmap	Specifies the bitmap to be written.
width *height*	Specify the width and height. These are the dimensions of the bitmap to be written.
x_hot *y_hot*	Specify where to place the hot spot coordinates (or –1,–1 if none present) in the file.

DESCRIPTION

XWriteBitmapFile writes a bitmap to a file. The file is written out in X version 11 bitmap format, which is the format created by the X version 11 *bitmap* program. Refer to that program's man pages for details. While *XReadBitmapFile* can read in either X version 10 format or X version 11 format, *XWriteBitmapFile* always writes out X version 11 format only. The difference between these is slight.

If the file cannot be opened for writing, *XWriteBitmapFile* returns *BitmapOpenFailed.* If insufficient memory is allocated *XWriteBitmapFile* returns *BitmapNoMemory.* Otherwise, on no error, *XWriteBitmapFile* returns *BitmapSuccess.*

If *x_hot* and *y_hot* are not –1, –1, then *XWriteBitmapFile* writes them out as the hot spot coordinates for the bitmap.

The following is an example of the contents of a bitmap file created. The name used ("gray" in this example) is the portion of *filename* after the last "/".

```
#define gray_width 16
#define gray_height 16
#define gray_x_hot 8
#define gray_y_hot 8
static char gray_bits[] =
  {
  0xf81f, 0xe3c7, 0xcff3, 0x9ff9,
  0xbffd, 0x33cc, 0x7ffe, 0x7ffe,
  0x7e7e, 0x7ffe, 0x37ec, 0xbbdd,
  0x9c39, 0xcff3, 0xe3c7, 0xf81f
  };
```

SEE ALSO

XSetTile, XQueryBestTile, XSetWindowBorderPixmap, XSetWindowBackgroundPixmap, XCreatePixmap, XCreatePixmapFromBitmapData, XFreePixmap, XQueryBestSize, XQueryBestStipple, XReadBitmapFile, XCreateBitmapFromData.

NAME

XXorRegion — calculate the difference between the union and intersection of two regions.

SYNOPSIS

```
XXorRegion(sra, srb, dr)
  Region sra, srb;
  Region  dr;                    /*  RETURN  */
```

ARGUMENTS

sra	
srb	Specify the two regions in which you want to perform the computation.
dr	Returns the result of the computation.

DESCRIPTION

XXorRegion calculates the union minus the intersection of two regions, and places it in *dr*. Xor is short for "Exclusive OR", meaning that a pixel is included in *dr* if it set in either *sra* or *srb* but not both.

STRUCTURES

```
/*
 * opaque reference to Regiondata type.
 * user won't need contents, only pointer.
 */
typedef struct _XRegion *Region;
```

SEE ALSO

XUnionRegion, XUnionRectWithRegion, XSubtractRegion, XShrinkRegion, XSetRegion, XRectInRegion, XPolygonRegion, XPointInRegion, XOffsetRegion, XIntersectRegion, XEmptyRegion, XCreateRegion, XDestroyRegion, XEqualRegion, XClipBox.

NAME

Xpermalloc — allocate memory never to be freed.

SYNOPSIS

```
char  *Xpermalloc(size)
        unsigned  int size;
```

ARGUMENTS

size	Specifies the size in bytes of the space to be allocated. This specification is rounded to the nearest 4-byte boundary.

DESCRIPTION

Xpermalloc allocates some memory that will not be freed until the process exits. *Xpermalloc* is used by some toolkits for permanently allocated storage and allows some performance and space savings over the completely general memory allocator.

NAME

XrmGetFileDatabase — retrieve a database from a file.

SYNOPSIS

```
XrmDatabase  XrmGetFileDatabase (filename)
         char  *filename;
```

ARGUMENTS

filename	Specifies the resource database file name.

DESCRIPTION

XrmGetFileDatabase opens the specified file, creates a new resource database, and loads it with the data read in from the file. The return value of the function is subsequently used to refer to the created database.

The specified file must contain lines in the format accepted by *XrmPutLineResource*. If it cannot open the specified file, *XrmGetFileDatabase* returns *NULL*.

For more information, refer to the *GSE Programmer's Guide*.

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.
```

SEE ALSO

XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmGetResource — get resource from name and class as strings.

SYNOPSIS

```
Bool XrmGetResource(database, str_name, str_class, str_type, value)
      XrmDatabase database;
      char *str_name;
      char *str_class;
      char **str_type;      /* RETURN */
      XrmValue *value;      /* RETURN */
```

ARGUMENTS

database	Specifies the database that is to be used.
str_name	Specifies the fully qualified name of the value being retrieved (as a string). *str_name* is an instance of a name retrieval key as described below.
str_class	Specifies the fully qualified class of the value being retrieved (as a string). *str_class* is an instance of a class retrieval key as described below.
str_type	Returns a pointer to the representation type of the destination. In this function, the representation type is itself represented as a string, not as an *XrmRepresentation.*
value	Returns the value in the database. Do not modify or free this data.

DESCRIPTION

The resource manager manages databases of resources consisting of lines containing resource name/class strings followed by a colon (:) and the value of the resource. *XrmGetResource* retrieves a resource from the specified database. It takes fully qualified name and class strings, and returns the representation and value of the matching resource. The value returned points into database memory; therefore, you must not modify that data. If a resource was found, *XrmGetResource* returns *True.* Otherwise, it returns *False.*

Currently, the database only frees or overwrites entries when new data is stored with *XrmMergeDatabases,* or *XrmPutResource* and related routines. A client that avoids these functions should be safe using the address passed back at any time until it exits.

XrmGetResource is very similar to *XrmQGetResource,* except that in *XrmQGetResource,* the equivalent argumnets to *str_name, str_class,* and *str_type* are quarks instead of strings.

To understand how data is stored and retrieved from the database, you must understand:

1) The basic components that make up the storage key and retrieval keys.

2) How keys are made up from components.

3) The two ways that components can be bound together.

4) What sort of keys are used to store and retrieve data.

5) How the storage key and retrieval keys are compared to determine whether they match.

6) If there are multiple matches, how the best match is chosen so only that corresponding value is returned.

Each will be covered in turn.

1) The storage key and retrieval keys are composed of a variable number of components bound together. There are two types of components: names and classes. By convention, names begin with a lower case character and classes begin with an upper case character. Therefore, **xmh, background,** and **toc** are examples of names, while **Xmh, Box,** and **Command** are examples of classes. A name key (like *str_name*) consists purely of names componenets. A class key (like *str_class*) consists purely of class components. The retrieval keys are a pair of keys, one composed of purely name components, the other of purely class components. A storage key (like *specifier* in *XrmPutResource*) consists of a mixture of name and class components.

2) A key is composed of multiple components bound together in sequence. This allows you to build logical keys for your application. For example, at the top level, the application might consist of a paned window (that is, a window divided into several sections) named **toc.** One pane of the paned window is a button box window named **buttons** filled with command buttons. One of these command buttons is used to retrieve (include) new mail and has the name **include.** This window has a fully qualified name **xmh.toc.buttons.include** and a fully qualified class **Xmh.VPaned.Box.Command.** Its fully qualified name is the name of its parent, **xmh.toc.buttons,** followed by its name **include.** Its class is the class of its parent, **Xmh.VPaned.Box,** followed by its particular class, **Command.**

3) The components in a key can be bound together in two ways: by a tight binding (a dot, .) or by a loose binding (an asterisk, *). Thus **xmh.toc.background** has three name components tightly bound together, while **Xmh*Command.foreground** uses both a loose and a tight binding. Bindings can also precede the first component (but may not follow the last component). By convention, if no binding is specified before the first component, a tight binding is assumed. For example, **xmh.background** and **.xmh.background** both begin with tight bindings before the **xmh**, while ***xmh.background** begins with a loose binding.

The difference between tight and loose bindings comes when comparing two keys. A tight binding means that the components on either side of the binding must be sequential. A loose binding is a sort of wildcard, meaning that there may be unspecified components between the two components that are loosely bound togehter. For example, **xmh.toc.background** would match **xmh*background** and ***background** but not **xmh.background** or **background.**

4) A key used to store data into the database can use both loose and tight bindings. This allows you to specify a data value which can match to many different retrieval keys. In contrast, keys used to retrieve data from the database can use only tight bindings. You can only look up one item in the database at a time. Remember also that a storage key can mix name and class components, while the retrieval keys are a pair of keys, one consisting purely of name (first character lower case) components and one consisting purely of class (capitalized) components.

5) The resource manager must solve the problem of how to compare the pair of retrieval keys to a single storage key. (Actually, to many single storage keys, since the resource manager will compare the retrieval keys against every key in the database, but one at a time.) The solution of comparing a pair of keys to a single key is simple. The resource manager compares component by component, comparing a component from the storage key against both the corresponding component from the name retrieval key, and the corresponding component from the class retrieval key. If the storage key component matches either retrieval key component, then that component is considered to match. For example, the storage key **xmh.toc.Foreground** matches the name key **xmh.toc.foreground** with the class key **Xmh.Box.Foreground.** This is why storage keys can mix name and class components, while retrieval keys cannot.

6) Because the resource manager allows loose bindings (wildcards) and mixing names and classes in the storage key, it is possible for many storage keys to match a single name/class retrieval key pair. To solve this problem, the resource manager uses the following precedence rules to determine which is the best match (and only the value from that match will be returned). The precedence rules are, in order of preference:

1. The attribute of the name and class must match. For example, queries for

```
xterm.scrollbar.background       (name)
XTerm.Scrollbar.Background       (class)
```

will not match the following database entry:

```
xterm.scrollbar:on
```

2. Database entries with name or class prefixed by a dot (.) are more specific than those prefixed by an asterisk (*). For example, the entry **xterm.geometry** is more specific than entry **xterm*geometry.**
3. Names are more specific than classes. For example, the entry ***scrollbar.background** is more specific than entry ***Scrollbar.Background.**
4. A name or class is more specific than omission. For example, the entry **Scrollbar*Background** is more specific than entry ***Background.**
5. Left components are more specific than right components. For example, **xterm*background** is more specific than entry **scrollbar*background.**

As an example of these rules, assume the following user preference specification:

```
xmh*background: red
*command.font:  8x13
*command.background:     blue
*Command.Foreground:     green
xmh.toc*Command.activeForeground:        black
```

A query for name **xmh.toc.messagefunctions.include.activeForeground** and class **Xmh.VPaned.Box.Command.Foreground** would match **xmh.toc*Command.activeForeground** and return black. However, it also matches ***Command.Foreground** but with lower preference, so it would not return green.

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef struct {
    unsigned int    size;
    caddr_t         addr;
} XrmValue;
```

SEE ALSO

XrmGetFileDatabase, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmGetStringDatabase — create a database from a string.

SYNOPSIS

```
XrmDatabase XrmGetStringDatabase(data)
    char *data;
```

ARGUMENTS

data	Specifies the database contents using a string.

DESCRIPTION

XrmGetStringDatabase creates a new database and stores in it the resources specified in *data*. The return value is subsequently used to refer to the created database. *XrmGetStringDatabase* is similar to *XrmGetFileDatabase,* except that it reads the information out of a string instead of a file. Each line is separated by a new line character in the format accepted by *XrmPutLineResource.*

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmInitialize — initialize the resource manager.

SYNOPSIS

```
void XrmInitialize();
```

DESCRIPTION

XrmInitialize initializes the resource manager, and should be called once before using any other resource manager functions. All it does is to create a representation type of "String" for values defined as strings. This representation type is used by *XrmPutStringResource* and *XrmQPutStringResource,* which require a value as a string. See *XrmQPutResource* for a description of representation types.

For more information refer to the *GSE Programmer's Guide.*

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmMergeDatabases — merge the contents of one database into another.

SYNOPSIS

void XrmMergeDatabases(*source_db*, *target_db***)**
  XrmDatabase *source_db*, ***target_db**;

ARGUMENTS

source_db	Specifies the descriptor of the resource database to be merged into the existing database.
target_db	Specifies a pointer to the descriptor of the resource database into which the *source_db* database will be merged.

DESCRIPTION

XrmMergeDatabases overwrites entries in the destination database. This procedure is used to combine databases, for example, an application specific database of defaults and a database of user preferences. The merge is destructive; it destroys the original *source_db* database and modifies the original *target_db*.

For more information refer to the *GSE Programmer's Guide*.

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmParseCommand — load resource data base from command line arguments.

SYNOPSIS

```
void XrmParseCommand(db, table, table_count, name, argc,
argv,)
            XrmDatabase *db;
            XrmOptionDescList table;
            int table_count;
            char *name;
            int *argc;          /* SEND and RETURN */
            char **argv;        /* SEND and RETURN */
```

ARGUMENTS

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*.
table	Specifies table of command line arguments to be parsed.
table_count	Specifies the number of entries in the table.
name	Specifies the application name.
argc	Before the call, specifies the number of arguments. After the call, returns the number of arguments not parsed.
argv	Before the call, specifies a pointer to the command line arguments. After the call, returns a pointer to a string containing the command line arguments that could not be parsed.

DESCRIPTION

XrmParseCommand parses an (*argc, argv*) pair according to the specified option table, loads recognized options into the specified database, and modifies the (*argc, argv*) pair to remove all recognized options.

The specified table is used to parse the command line. Recognized entries in the table are removed from *argv*, and entries are made in the specified resource database. The table entries contain information on the option string, the option name, which style of option and a value to provide if the option kind is *XrmoptionNoArg*. See the example table below.

argc specifies the number of arguments in *argv* and is set to the remaining number of arguments that were not parsed. *name* should be the name of your application for use in building the data base entry. *name* is prepended to the *resourceName* in the option table before storing the specification. No separating (binding) character is inserted. The table must contain either a dot (".") or an asterisk ("*") as the first character in the *resourceName* entry. The *resourceName* entry can contain multiple components.

The following is a typical options table:

```
static XrmOptionDescRec opTable[] = {
{"-background",   "*background",                  XrmoptionSepArg, (caddr_t) NULL},
{"-bd",           "*borderColor",                 XrmoptionSepArg, (caddr_t) NULL},
{"-bg",           "*background",                  XrmoptionSepArg, (caddr_t) NULL},
{"-borderwidth", "*TopLevelShell.borderWidth",    XrmoptionSepArg, (caddr_t) NULL},
{"-bordercolor", "*borderColor",                  XrmoptionSepArg, (caddr_t) NULL},
{"-bw",           "*TopLevelShell.borderWidth",   XrmoptionSepArg, (caddr_t) NULL},
{"-display",      ".display",                     XrmoptionSepArg, (caddr_t) NULL},
{"-fg",           "*foreground",                  XrmoptionSepArg, (caddr_t) NULL},
{"-fn",           "*font",                        XrmoptionSepArg, (caddr_t) NULL},
{"-font",         "*font",                        XrmoptionSepArg, (caddr_t) NULL},
{"-foreground",   "*foreground",                  XrmoptionSepArg, (caddr_t) NULL},
{"-geometry",     ".TopLevelShell.geometry",      XrmoptionSepArg, (caddr_t) NULL},
{"-iconic",       ".TopLevelShell.iconic",        XrmoptionNoArg,  (caddr_t) "on"},
{"-name",         ".name",                        XrmoptionSepArg, (caddr_t) NULL},
{"-reverse",      "*reverseVideo",                XrmoptionNoArg,  (caddr_t) "on"},
{"-rv",           "*reverseVideo",                XrmoptionNoArg,  (caddr_t) "on"},
{"-synchronous", ".synchronous",                  XrmoptionNoArg,  (caddr_t) "on"},
{"-title",        ".TopLevelShell.title",         XrmoptionSepArg, (caddr_t) NULL},
{"-xrm",          NULL,                           XrmoptionResArg, (caddr_t) NULL},
};
```

In this table, if the –background (or –bg) option is used to set background colors, the stored resource specifier will match all resources of attribute background. If the –borderwidth option is used, the stored resource specifier applies only to border width attributes of class TopLevelShell (that is, outer-most windows, including pop-up windows). If the –title option is used to set a window name, only the top-most application windows receive the resource.

When parsing the command line, any unique unambiguous abbreviation for an option name in the table is considered a match for the option.

For more information, refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef enum {
  XrmoptionNoArg,     /* Value is specified in OptionDescRec.value */
  XrmoptionIsArg,     /* Value is the option string itself */
  XrmoptionStickyArg,/* Value is chars immediately following option*/
  XrmoptionSepArg,    /* Value is next argument in argv */
  XrmoptionResArg,    /* Resource & value in next argument in argv */
  XrmoptionSkipArg,   /* Ignore this option & next argument in argv */
  XrmoptionSkipLine   /* Ignore this option & the rest of argv */
} XrmOptionKind;

typedef struct {
  char *option;        /* Option specification string in argv */
  char *resourceName; /*Binding & resource name (sans application name)
  XrmOptionKind argKind; /* Which style of option it is */
  caddr_t value;       /* Value to provide if XrmoptionNoArg */
} XrmOptionDescRec, *XrmOptionDescList;
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmPutFileDatabase — store database in file.

SYNOPSIS

```
void XrmPutFileDatabase(database, stored_db)
        XrmDatabase database;
        char *stored_db;
```

ARGUMENTS

database	Specifies the database that is to be saved.
stored_db	Specifies the file name for the stored database.

DESCRIPTION

XrmPutFileDatabase stores a copy of the application's current database in the specified file. The file is an ASCII text file that contains lines in the format that is accepted by *XrmPutLineResource.*

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmPutLineResource — add resource entry given as string of name and value.

SYNOPSIS

```
void XrmPutLineResource(database, line)
     XrmDatabase *database; /* SEND and, if NULL,
RETURN */
     char *line;
```

ARGUMENTS

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 name and value pair as a single string, in the format *resource:value.* A single colon (:) separates the resource name from the value, for example, "xterm*background:green\n".

DESCRIPTION

XrmPutLineResource adds a single resource entry to the specified database.

XrmPutLineResource is similar to *XrmPutStringResource,* except that instead of having separate string arguments for the resource and its value, *XrmPutLineResource* takes a single string argument (*line*) which consists of the resource name, a colon, then the value. Since the value is a string, it is stored into the database with representation type "String."

Any whitespace before or after the name or colon in the *line* argument is ignored. The value is terminated by a new-line or a *NULL* character. The value may contain embedded new-line characters represented by the "\" and "n" two character pair (not the single "\n" character), which are converted into a single linefeed character. In addition, the value may run over onto the next line, this is indicated by a "\" character at the end of the line immediately preceding the "\n" character.

NULL teminated strings without a new line are also permitted. *XrmPutResource, XrmQPutResource, XrmPutStringResource, XrmQPutStringResource* and *XrmPutLineResource* all store data into a database. See *XrmQPutResource* for the most complete description of this process.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

`XrmDatabase is a pointer to an opaque data type.`

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmPutResource — store a resource into a database.

SYNOPSIS

```
void XrmPutResource(database, specifier, type, value)
        XrmDatabase *database; /* SEND and, if NULL,
RETURN */
        char *specifier;
        char *type;
        XrmValue *value;
```

ARGUMENTS

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*.
specifier	Specifies a partial specification of the resource.
type	Specifies the type of the resource.
value	Specifies the value of the resource.

DESCRIPTION

XrmPutResource is one of several functions which store data into a database.

XrmQPutResource first converts *specifier* into a binding list and a quark list by calling *XrmStringToBindingQuarkList,* and converts *type* into an *XrmRepresentation* by calling *XrmStringToRepresentation.* Finally, it puts the data into the database.

XrmPutResource, XrmQPutResource, XrmPutStringResource, XrmQPutStringResource and *XrmPutLineResource* all store data into a database. See *XrmQPutResource* for the most complete description of this process.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef struct {
    unsigned int   size;
    caddr_t        addr;
} XrmValue, *XrmValuePtr;
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmPutStringResource — add a resource that is specified as a string.

SYNOPSIS

```
void XrmPutStringResource(database, resource, value)
     XrmDatabase *database; /* SEND, and if NULL,
RETURN */
     char *resource;
     char *value;
```

ARGUMENTS

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.*
resource	Specifies the resource as a string.
value	Specifies the value of the resource. The value is specified as a string.

DESCRIPTION

XrmPutStringResource adds a resource with the specified value to the specified database. The *resource* string may contain both names and classes, bound with either loose (*) or tight (.) bindings. See *XrmGetResource* for more information about bindings.

The representation type used in the database is "String."

XrmPutResource, XrmQPutResource, XrmPutStringResource, XrmQPutStringResource and *XrmPutLineResource* all store data into a database. See *XrmQPutResource* for the most complete description of this process.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmQGetResource — get resource from name and class as quarks.

SYNOPSIS

```
Bool XrmQGetResource(database, quark_name, quark_class,
quark_type, value)
      XrmDatabase database;
      XrmNameList quark_name;
      XrmClassList quark_class;
      XrmRepresentation *quark_type; /* RETURN */
      XrmValue *value;      /* RETURN */
```

ARGUMENTS

database	Specifies the database that is to be used.
quark_name	Specifies the fully qualified name of the value being retrieved (as a list of quarks).
quark_class	Specifies the fully qualified class of the value being retrieved (as a list of quarks).
quark_type	Returns a pointer to the representation type of the destination. In this function, the representation type is itself represented as a quark.
value	Returns a pointer to the value in the database. Do not modify or free this data.

DESCRIPTION

XrmQGetResource retrieves a resource from the specified database. It takes fully qualified name and class strings, and returns the representation and value of the matching resource. The value returned points into database memory; therefore, you must not modify that data. If a resource was found, *XrmQGetResource* returns *True*. Otherwise, it returns *False*.

Currently, the database only frees or overwrites entries when new data is stored with *XrmMergeDatabases,* or *XrmPutResource* and related routines. A client that avoids these functions should be safe using the address passed back at any time until it exits.

XrmQGetResource is very similar to *XrmGetResource,* except that in *XrmGetResource,* the equivalent arguments to *quark_name, quark_class,* and *quark_type* arguments are strings instead of quarks.

Refer to *XrmGetResource* for a description of how data is looked up in the database. Refer also to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef XrmQuarkList XrmNameList;
typedef XrmQuarkList XrmClassList;
typedef XrmQuark     XrmRepresentation;

typedef struct {
    unsigned int    size;
    caddr_t         addr;
} XrmValue, *XrmValuePtr;
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmQGetSearchList — return a list of database levels.

SYNOPSIS

```
Bool XrmQGetSearchList(database, names, classes, search_list,
list_length)
        XrmDatabase database;
        XrmNameList names;
        XrmClassList classes;
        XrmSearchList search_list;  /* RETURN */
        int list_length;
```

ARGUMENTS

database	Specifies the database that is to be used.
names	Specifies a list of resource names.
classes	Specifies a list of resource classes.
search_list	Returns a search list for further use. The caller must allocate sufficient space for the list before calling *XrmQGetSearchList.*
list_length	Specifies the number of entries (not the byte size) allocated for list.

DESCRIPTION

XrmQGetSearchList is a tool for searching the database more efficiently. It is used in combination with *XrmGetSearchResource.* Often, one searches the database for many similar resources which differ only in their final component (e.g., **xmh.toc.foreground**, **xmh.toc.background**, etc). Rather than looking for each resource in its entirety, *XrmGetSearchList* searches the database for the common part of the resource name, returning a whole list of items in the database that match it. This list is called the "search list". This search list is then used by *XrmQGetSearchList,* which searches for the last components one at a time. In this way, the common work of searching for similar resources is done only once, and the specific part of the search is done on the much shorter search list.

XrmQGetSearchList takes a list of names and classes and returns a list of database levels where a match might occur. The returned list is in best-to-worst order and uses the same algorithm as *XrmGetResource* for determining precedence. If *search_list* was large enough for the search list, *XrmQGetSearchList* returns *True.* Otherwise, it returns *False.*

The size of the search list that must be allocated by the caller is dependent upon the number of levels and wildcards in the resource specifiers that are stored in the database. The worst case length is 3^n, where n is the number of name or class components in *names* or *classes*.

Only the common prefix of a resource name should be specified in the name and class list to *XrmQGetSearchList*. In the example above, the common prefix would be **xmh.toc**. However, note that *XrmQGetSearchResource* requires that *name* represent a single component only. Therefore, the common prefix must be all but the last componenet of the name and class.

For more information, refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef XrmQuarkList XrmNameList;
typedef XrmQuarkList XrmClassList;
typedef XrmQuark     XrmRepresentation;

XrmSearchList is a pointer to an opaque data type.
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmQGetSearchResource — search resource database levels for a given resource.

SYNOPSIS

```
Bool XrmQGetSearchResource(search_list, name, class, type,
value)
          XrmSearchList search_list;
          XrmName name;
          XrmClass class;
          XrmRepresentation *type;/* RETURN */
          XrmValue *value;   /* RETURN */
```

ARGUMENTS

search_list	Specifies the search list returned by *XrmQGetSearchList.*
name	Specifies the resource name.
class	Specifies the resource class.
type	Returns data representation type.
value	Returns the value in the database.

DESCRIPTION

XrmQGetSearchResource is a tool for searching the database more efficiently. It is used in combination with *XrmGetSearchList.* Often, one searches the database for many similar resources which differ only in their final component (e.g., **xmh.toc.foreground**, **xmh.toc.background**, etc). Rather than looking for each resource in its entirety, *XrmGetSearchList* searches the database for the common part of the resource name, returning a whole list of items in the database that match it. This list is called the "search list". *XrmQGetSearchResource* searches the search list for the resource that is fully identified by *name* and *class.* The search stops with the first match. *XrmQGetSearchResource* returns *True* if the resource was found.

A call to *XrmQGetSearchList* with a name and class list containing all but the last component of a resource name followed by a call to *XrmQGetSearchResource* with the last component name and class returns the same database entry as *XrmGetResource* or *XrmQGetResource* would with the fully qualified name and class.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef XrmQuark XrmName;
typedef XrmQuark XrmClass;
typedef XrmQuark XrmRepresentation;

typedef struct {
    unsigned int    size;
    caddr_t         addr;
} XrmValue, *XrmValuePtr;

XrmSearchList is a pointer to an opaque data type.
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmQPutResource — store a resource into a database.

SYNOPSIS

```
void XrmQPutResource(database, bindings, quarks, type, value)
    XrmDatabase *database;  /* SEND and, if NULL,
RETURN */
    XrmBindingList bindings;
    XrmQuarkList quarks;
    XrmRepresentation type;
    XrmValue *value;
```

ARGUMENTS

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.*
bindings	Specifies a list of bindings for binding together the *quarks* argument.
quarks	Specifies the partial name or class list of the resource to be stored.
type	Specifies the type of the resource.
value	Specifies the value of the resource.

DESCRIPTION

XrmQPutResource stores a resource into the database.

database can be a previously defined database, as returned by *XrmGetStringDatabase, XrmGetFileDatabase,* or from *XrmMergeDatabases.* If *database* is *NULL,* a new database is created and a pointer to it returned in *database.*

bindings and *quarks* together specify where the value should be stored in the database. See *XrmStringToBindingQuarkList* for a brief description of binding and quark lists. See *XrmGetResource* for a lengthy description of the resource managaer naming conventions and lookup rules.

type is the representation type of *value.* This provides a way to distinguish between different representations of the same information. Representation types are user defined character strings describing the way the data is represented. For example, a color may be specified by a color name ("red"), or be coded in a hexadecimal string ("#4f6c84") (if it is to be used as an argument to *XParseColor.*) The representation type would distinguish between these two. Representation types are created from simple character strings by using the macro *XrmStringToRepresentation*. The type *XrmRepresentation* is actually the same type as *XrmQuark*, since it is an ID for a string. The representation is stored along with the value in the database, and is returned when the database is accessed.

value is the value of the resource, specified as an *XrmValue*.

XrmGetResource contains the complete description of how data is accessed from the database, and so provides a good perspective on how it is stored.

For more information, refer to the *GSE Programmer's Guide*.

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef enum{XrmBindTightly,XrmBindLoosely}XrmBinding,*XrmBindingList;

typedef int                      XrmQuark, *XrmQuarkList;
typedef XrmQuarkList             XrmNameList;
typedef XrmQuark                 XrmRepresentation;

typedef struct {
    unsigned int    size;
    caddr_t         addr;
} XrmValue, *XrmValuePtr;
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmQPutStringResource — add a string resource value to database using quarks.

SYNOPSIS

```
void XrmQPutStringResource(database, bindings, quarks, value)
     XrmDatabase *database; /* SEND, and if NULL,
RETURN */
     XrmBindingList bindings;
     XrmQuarkList quarks;
     char *value;
```

ARGUMENTS

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*.
bindings	Specifies a list of bindings for binding together the *quarks* argument.
quarks	Specifies the partial name or class list of the resource to be stored.
value	Specifies the value of the resource as a string.

DESCRIPTION

XrmQPutStringResource stores a resource into the specified database.

XrmQPutStringResource is a cross between *XrmQPutResource* and *XrmPutStringResource.* Like *XrmQPutResource,* it specifies the resource by *quarks* and *bindings,* two lists that together make a name/class list with loose and tight bindings. Like *XrmPutStringResource,* it specifies the value to be stored as a string, that value is converted into an *XrmValue,* and the default "String" representation type is used.

XrmPutResource, *XrmQPutResource,* *XrmPutStringResource,* *XrmQPutStringResource* and *XrmPutLineResource* all store data into a database. See *XrmQPutResource* for the most complete description of this process.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
XrmDatabase is a pointer to an opaque data type.

typedef enum{XrmBindTightly,XrmBindLoosely}XrmBinding,*XrmBindingList

typedef int     XrmQuark, *XrmQuarkList;
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmQuarkToString — convert a quark to a string.

SYNOPSIS

```
char  *XrmQuarkToString(quark)
          XrmQuark quark;
```

ARGUMENTS

quark	Specifies the quark for which the equivalent string is desired.

DESCRIPTION

XrmQuarkToString returns the string for which the quark is serving as a shorthand symbol. The quark was earlier set to represent the string by *XrmStringToQuark*. The string pointed to by the return value must not be modified or freed because that string is in the data structure used by the resource manager for assigning quarks.

Quarks are used by the resource manager to represent strings. The resource manager needs to make many comparisons of strings when it gets data from the database. It is more efficient for the resource manager to convert these strings into quarks, and to compare quarks instead. Since quarks are presently represented by integers, comparing quarks is trivial.

The three #define statements in the structures section provide an extra level of abstraction. They define macros so that names, classes, and representations can also be represented as quarks.

For more information, refer to the *GSE Programmer's Guide.*

STRUCTURES

```
typedef int     XrmQuark;

/* macro definitions from <X11/resource.h> */

#define XrmNameToString(name) XrmQuarkToString(name)
#define XrmClassToString(class) XrmQuarkToString(class)
#define XrmRepresentationToString(type) XrmQuarkToString(type)
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmStringToBindingQuarkList — convert key string to binding list and quark list.

SYNOPSIS

XrmStringToBindingQuarkList(*string*, *bindings*, *quarks*)
        char **string*;
        XrmBindingList *bindings*; /* RETURN */
        XrmQuarkList *quarks*; /* RETURN */

ARGUMENTS

string	Specifies the string for which the list of quarks and list of bindings are to be generated. Must be *NULL* terminated.
bindings	Returns the binding list. The caller must allocate sufficient space for the binding list before the call.
quark	Returns the list of quarks. The caller must allocate sufficient space for the quarks list before the call.

DESCRIPTION

XrmStringToBindingQuarkList converts the string into two lists - one of quarks and one of bindings. Component names in the list are separated by a dot (.) indicating a tight binding or an asterisk (*) indicating a loose binding. If the string does not start with dot or asterisk, a dot (.) is assumed.

A tight binding means that the quarks on either side of the binding are consecutive in the key. A loose binding, on the other hand, is a wildcard which can match any number of unspecified components in between the two quarks separated by the binding. Tight and loose bindings are used in the match rules, which compare multi-component strings to find matches and determine the best match. See *XrmGetResource* for a full description of lookup rules.

For example, **"*a.b*c"** becomes:

quarks	"a"	"b"	"c"
bindings	**XrmBindLoosely**	**XrmBindTightly**	**XrmBindLoosely**

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
typedef int XrmQuark, *XrmQuarkList;
typedef enum(XrmBindLoosely,XrmBindTightly)XrmBinding,*XrmBindingList
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmStringToQuark — convert a string to a quark.

SYNOPSIS

```
XrmQuark  XrmStringToQuark(string)
          char  *string;
```

ARGUMENTS

string	Specifies the string for which a quark is to be allocated.

DESCRIPTION

XrmStringToQuark returns a quark that is equivalent to the specified string. If a quark already exists for the string, that previously existing quark is returned. If no quark exists for the string, then a new quark is created, assigned to the string, and *string* is copied into the quark table. (Since *string* is copied, it may be freed. However, the copy of the string in the quark table must not be modified or freed.) *XrmQuarkToString* performs the inverse function.

Quarks are used by the resource manager to represent strings. The resource manager needs to make many comparisons of strings when it gets data from the database. It is more efficient for the resource manager to convert these strings into quarks and to compare quarks instead. Since quarks are presently represented by integers, comparing quarks is trivial.

The three #define statements in the structures section provide an extra level of abstraction. They define macros so that names, classes and representations can also be represented as quarks.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
typedef int                    XrmQuark;
```

```
/* macro definitions from <X11/resource.h> */

#define XrmStringToName(string) XrmStringToQuark(string)
#define XrmStringToClass(string) XrmStringToQuark(string)
#define XrmStringToRepresentation(string) XrmStringToQuark(string)
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmUniqueQuark.

NAME

XrmStringToQuarkList — convert key string to quark list.

SYNOPSIS

```
void XrmStringToQuarkList(string, quarks)
        char *string;
        XrmQuarkList quarks;/* RETURN */
```

ARGUMENTS

string	Specifies the string for which a list of quarks is to be generated. Must be *NULL* terminated. The components must be separated by the "." character (tight binding), not "*" characters (loose bindings).
quarks	Returns the list of quarks.

DESCRIPTION

XrmStringToQuarkList converts *string* (generally a fully qualified name/class string) to a list of quarks. Components of the string must be separated by a tight binding (the "." character). Use *XrmStringToBindingQuarkList* for lists which contain both tight and loose bindings. Refer to *XrmGetResource* for a description of tight and loose binding.

Each component of the string is individually converted into a quark. Refer to *XrmStringToQuark* for information about quarks and converting strings to quarks. *quarks* is a *NULL* terminated list of quarks.

For example, **xmh.toc.command.background** is converted into a list of four quarks: the quarks for **xmh, toc, command,** and **background,** in that order. A *NULLQUARK* is appended to the end of the list.

Note that *XrmStringToNameList* and *XrmStringToClassList* are macros that perform exactly the same function as *XrmStringToQuarkList*. These may be used in cases where they clarify the code.

For more information refer to the *GSE Programmer's Guide*.

STRUCTURES

```
typedef int XrmQuark *XrmQuarkList;

#define XrmStringToNameList(str, name)XrmStringToQuarkList((str),(name))
#define XrmStringToClassList(str,class)XrmStringToQuarkList((str),(class))
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuark, XrmUniqueQuark.

NAME

XrmUniqueQuark — allocate a new quark.

SYNOPSIS

```
XrmQuark  XrmUniqueQuark()
```

DESCRIPTION

XrmUniqueQuark allocates a quark that is guaranteed not to represent any existing string. For most applications, *XrmStringToQuark* is more useful, as it binds a quark to a string. However, on some occasions, you may want to allocate a quark that has no string equivalent.

The shorthand name for a string is called a "quark" and is the type *XrmQuark*. Quarks are used to improve performance of the resource manager, which must make many string comparisons. Quarks are presently represented as `ints`. Simple comparisons of quarks can be performed rather than lengthy string comparisons.

A quark is to a string what an atom is to a property name in the server, but its use is entirely local to your application.

For more information refer to the *GSE Programmer's Guide.*

STRUCTURES

```
typedef int XrmQuark;
```

SEE ALSO

XrmGetFileDatabase, XrmGetResource, XrmGetStringDatabase, XrmInitialize, XrmMergeDatabases, XrmParseCommand, XrmPutFileDatabase, XrmPutLineResource, XrmPutResource, XrmPutStringResource, XrmQGetResource, XrmQGetSearchList, XrmQGetSearchResource, XrmQPutResource, XrmQPutStringResource, XrmQuarkToString, XrmStringToBindingQuarkList, XrmStringToQuarkList, XrmStringToQuark.

RESOURCE MANAGER

3X

PERMUTED INDEX

for X root	window parameter setting utility	xsetroot(1)
/change pointer	acceleration	XChangePointerControl(3X)
/get current pointer	acceleration parameters	XGetPointerControl(3X)
/disable or enable	access control	XSetAccessControl(3X)
XAddHost: add a host to the	access control list	XAddHost(3X)
add multiple hosts to the	access control list XAddHosts:	XAddHosts(3X)
/enable changes to the	access control list	XEnableAccessControl(3X)
XRemoveHost: remove host from	access control list	XRemoveHost(3X)
/remove multiple hosts from the	access control list	XRemoveHosts(3X)
/prevent modification to the host	access list	XDisableAccessControl(3X)
/obtain a list of hosts having	access to this display	XListHosts(3X)
XActivateScreenSaver:	activate screen blanking	XActivateScreenSaver(3X)
/change parameters of	active pointer grab	XChangeActivePointerGrab(3X)
list XAddHost:	add a host to the access control	XAddHost(3X)
XInsertModifiermapEntry:	add a new entry to an/	XInsertModifiermapEntry(3X)
as a/ XrmPutStringResource:	add a resource that is specified	XrmPutStringResource(3X)
database/ XrmQPutStringResource:	add a string resource value to	XrmQPutStringResource(3X)
value in image XAddPixel:	add constant value to every pixel	XAddPixel(3X)
control list XAddHosts:	add multiple hosts to the access	XAddHosts(3X)
the client's/ XChangeSaveSet:	add or remove a subwindow from	XChangeSaveSet(3X)
XUnionRectWithRegion:	add rectangle to region	XUnionRectWithRegion(3X)
string of/ XrmPutLineResource:	add resource entry given as	XrmPutLineResource(3X)
save-set XAddToSaveSet:	add window's children to client's	XAddToSaveSet(3X)
XrmUniqueQuark:	allocate a new quark	XrmUniqueQuark(3X)
cell with closest/ XAllocColor:	allocate a read-only colormap	XAllocColor(3X)
structure XCreateImage:	allocate memory for an XImage	XCreateImage(3X)
Xpermalloc:	allocate memory never to be freed	Xpermalloc(3X)
color name XAllocNamedColor:	allocate read-only colorcell from	XAllocNamedColor(3X)
color name XStoreNamedColor:	allocate read/write colorcell by	XStoreNamedColor(3X)
XAllocColorPlanes:	allocate read/write/	XAllocColorPlanes(3X)
colorcells XAllocColorCells:	allocate read/write (non-shared)	XAllocColorCells(3X)
XFreeFontPath: free memory	allocated by XGetFontPath	XFreeFontPath(3X)
/free the memory	allocated for association table	XDestroyAssocTable(3X)
XFreeExtensionList: free memory	allocated for list of installed/	XFreeExtensionList(3X)
	analog / digital clock for X	xclock(1)
XFillArc: fill an	arc	XFillArc(3X)
XDrawArc: draws an	arc fitting inside a rectangle	XDrawArc(3X)
XSetArcMode: set	arc mode in graphics context	XSetArcMode(3X)

XReadBitmapFile: read a	bitmap from disk	XReadBitmapFile(3X)
XWriteBitmapFile: write	bitmap to file	XWriteBitmapFile(3X)
create a cursor from two	bitmaps XCreatePixmapCursor:	XCreatePixmapCursor(3X)
XCopyPlane: copy and color	bit-plane of drawable	XCopyPlane(3X)
graphics/ XSetFunction: set	bitwise logical operation in	XSetFunction(3X)
activate screen	blanking XActivateScreenSaver:	XActivateScreenSaver(3X)
XSetWindowBorder: change window	border attribute to pixel value	XSetWindowBorder(3X)
/change window	border tile attribute	XSetWindowBorderPixmap(3X)
(SetWindowBorderWidth: change the	border width of a window	XSetWindowBorderWidth(3X)
/change window position, size,	border width, or stacking order	XConfigureWindow(3X)
order /circulate	bottom child to top of stacking	XCirculateSubwindowsDown(3X)
/circulate top child to	bottom of stacking order	XCirculateSubwindowsUp(3X)
XFetchBytes: return data from cut	buffer 0	XFetchBytes(3X)
XStoreBytes: store data in cut	buffer 0	XStoreBytes(3X)
return data from cut	buffer XFetchBuffer:	XFetchBuffer(3X)
XStoreBuffer: store data in a cut	buffer	XStoreBuffer(3X)
XPending: flush the output	buffer and return the number of/	XPending(3X)
and errors/ XSync: flush output	buffer and wait for all events	XSync(3X)
XFlush: flush the output	buffer (display all queued/	XFlush(3X)
XRotateBuffers: rotate the cut	buffers	XRotateBuffers(3X)
XGrabButton: grab a pointer	button	XGrabButton(3X)
XUngrabButton: release a	button from grab	XUngrabButton(3X)
/get the pointer	button mapping	XGetPointerMapping(3X)
XSetPointerMapping: set pointer	button mapping	XSetPointerMapping(3X)
the union and/ XXorRegion:	calculate the difference between	XXorRegion(3X)
user geometry string/ XGeometry:	calculate window geometry given	XGeometry(3X)
XSetAfterFunction: set function	called after all Xli	XSetAfterFunction(3X)
allocate a read-only colormap	cell with closest/ XAllocColor:	XAllocColor(3X)
XFreeColors: free colormap	cells or planes	XFreeColors(3X)
a window XChangeProperty:	change a property associated with	XChangeProperty(3X)
XReparentWindow:	change a window's parent	XReparentWindow(3X)
XResizeWindow:	change a window's size	XResizeWindow(3X)
context to/ XSetClipRectangles:	change clip_mask in graphics	XSetClipRectangles(3X)
XSetCloseDownMode:	change close down mode of client	XSetCloseDownMode(3X)
XRecolorCursor:	change color of cursor	XRecolorCursor(3X)
context XChangeGC:	change components of a graphics	XChangeGC(3X)
window to/ XTranslateCoordinates:	change coordinate system from one	XTranslateCoordinates(3X)
XChangeKeyboardMapping:	change keyboard mapping	XChangeKeyboardMapping(3X)
as key/ XChangeKeyboardControl:	change keyboard preferences such	XChangeKeyboardControl(3X)
XOffsetRegion:	change offset of region	XOffsetRegion(3X)
XChangeActivePointerGrab:	change parameters of active/	XChangeActivePointerGrab(3X)
XChangePointerControl:	change pointer acceleration	XChangePointerControl(3X)

/load resource data base from	command line arguments	XrmParseCommand(3X)
/set line drawing	components in graphics context	XSetLineAttributes(3X)
XChangeGC: change	components of a graphics context	XChangeGC(3X)
regions XIntersectRegion:	compute the intersection of two	XIntersectRegion(3X)
XUnionRegion:	compute the union of two regions	XUnionRegion(3X)
server XOpenDisplay:	connect a client program to an X	XOpenDisplay(3X)
XDrawLines: draw multiple	connected lines	XDrawLines(3X)
/reports the display name when	connecting to that display fails	XDisplayName(3X)
XNoOp: send a NoOp to exercise	connection with server	XNoOp(3X)
value in image XAddPixel: add	constant value to every pixel	XAddPixel(3X)
XrmMergeDatabases: merge the	contents of one database into/	XrmMergeDatabases(3X)
drawable into/ XGetImage: place	contents of rectangle from	XGetImage(3X)
XUniqueContext: create a new	context ID (not graphics context)	XUniqueContext(3X)
change components of a graphics	context XChangeGC:	XChangeGC(3X)
XCopyGC: copy a graphics	context	XCopyGC(3X)
context manager (not graphics	context) /get data from	XFindContext(3X)
XFreeGC: free a graphics	context	XFreeGC(3X)
and context type (not graphics	context) /corresponding to window	XSaveContext(3X)
set arc mode in graphics	context XSetArcMode:	XSetArcMode(3X)
pixel value in graphics	context /set background	XSetBackground(3X)
set clip_mask pixmap in graphics	context XSetClipMask:	XSetClipMask(3X)
set clip origin in graphics	context XSetClipOrigin:	XSetClipOrigin(3X)
set fill rule in graphics	context XSetFillRule:	XSetFillRule(3X)
set fill style in graphics	context XSetFillStyle:	XSetFillStyle(3X)
set current font in graphics	context XSetFont:	XSetFont(3X)
pixel value in graphics	context /set foreground	XSetForeground(3X)
logical operation in graphics	context /set bitwise	XSetFunction(3X)
flag in graphics	context /set graphics-exposures	XSetGraphicsExposures(3X)
drawing components in graphics	context /set line	XSetLineAttributes(3X)
set plane mask in graphics	context XSetPlaneMask:	XSetPlaneMask(3X)
set stipple in graphics	context XSetStipple:	XSetStipple(3X)
set subwindow mode in graphics	context XSetSubwindowMode:	XSetSubwindowMode(3X)
tile/stipple origin in graphics	context XSetTSOrigin: set	XSetTSOrigin(3X)
set fill tile in graphics	context XSetTile:	XSetTile(3X)
a new context ID (not graphics	context) XUniqueContext: create	XUniqueContext(3X)
and type XDeleteContext: delete	context entry for given window	XDeleteContext(3X)
XCreateGC: create new graphics	context for a drawable	XCreateGC(3X)
XFindContext: get data from	context manager (not graphics/	XFindContext(3X)
/change clip_mask in graphics	context to list of rectangles	XSetClipRectangles(3X)
/set the clip_mask of the graphics	context to the specified region	XSetRegion(3X)
value corresponding to window and	context type (not graphics/ /data	XSaveContext(3X)
obtain list of current keyboard	control XGetKeyboardControl:	XGetKeyboardControl(3X)

disable or enable access	control XSetAccessControl:	XSetAccessControl(3X)
add a host to the access	control list XAddHost:	XAddHost(3X)
add multiple hosts to the access	control list XAddHosts:	XAddHosts(3X)
/enable changes to the access	control list	XEnableAccessControl(3X)
remove host from access	control list XRemoveHost:	XRemoveHost(3X)
multiple hosts from the access	control list /remove	XRemoveHosts(3X)
and pointer events/ XAllowEvents:	control the behavior of keyboard	XAllowEvents(3X)
appropriate/ XKeysymToKeycode:	convert a keysym to the	XKeysymToKeycode(3X)
XrmQuarkToString:	convert a quark to a string	XrmQuarkToString(3X)
XrmStringToQuark:	convert a string to a quark	XrmStringToQuark(3X)
ASCII XKeysymToString:	convert *KeySym* symbol to	XKeysymToString(3X)
XKeycodeToKeysym:	convert key code to keysym	XKeycodeToKeysym(3X)
XrmStringToBindingQuarkList:	convert key string to binding/	XrmStringToBindingQuarkList(3X)
XrmStringToQuarkList:	convert key string to quark list	XrmStringToQuarkList(3X)
code XStringToKeysym:	convert keysym name to keysym	XStringToKeysym(3X)
to/ XTranslateCoordinates: change	coordinate system from one window	XTranslateCoordinates(3X)
olormap/ XCopyColormapAndFree:	copy a colormap and return new	XCopyColormapAndFree(3X)
XCopyGC:	copy a graphics context	XCopyGC(3X)
XCopyArea:	copy an area of a drawable	XCopyArea(3X)
drawable XCopyPlane:	copy and color bit-plane of	XCopyPlane(3X)
location within/ XGetSubImage:	copy rectangle in drawable to	XGetSubImage(3X)
XCreateColormap:	create a colormap	XCreateColormap(3X)
XCreateGlyphCursor:	create a cursor from font glyphs	XCreateGlyphCursor(3X)
cursor font XCreateFontCursor:	create a cursor from standard	XCreateFontCursor(3X)
XCreatePixmapCursor:	create a cursor from two bitmaps	XCreatePixmapCursor(3X)
XrmGetStringDatabase:	create a database from a string	XrmGetStringDatabase(3X)
mapping/ XNewModifiermap:	create a keyboard modifier	XNewModifiermap(3X)
(X10) XCreateAssocTable:	create a new association table	XCreateAssocTable(3X)
graphics/ XUniqueContext:	create a new context ID (not	XUniqueContext(3X)
XCreateRegion:	create a new empty region	XCreateRegion(3X)
XCreatePixmap:	create a pixmap	XCreatePixmap(3X)
XCreatePixmapFromBitmapData:	create a pixmap with depth from/	XCreatePixmapFromBitmapData(3X)
XSetStandardColormap:	create a standard colormap	XSetStandardColormap(3X)
attributes XCreateWindow:	create a window and set	XCreateWindow(3X)
format/ XCreateBitmapFromData:	create bitmap from X11 bitmap	XCreateBitmapFromData(3X)
table XMakeAssoc:	create entry in an association	XMakeAssoc(3X)
drawable XCreateGC:	create new graphics context for a	XCreateGC(3X)
image XSubImage:	create subimage from part of	XSubImage(3X)
/free specified in-memory data	created by an Xlib function	XFree(3X)
XCreateSimpleWindow:	creates an unmapped/	XCreateSimpleWindow(3X)
XGetWindowAttributes: obtain	current attributes of window	XGetWindowAttributes(3X)
XSetFont: set	current font in graphics context	XSetFont(3X)

XGetFontPath: get the	current font search path	XGetFontPath(3X)
XGetGeometry: obtain	current geometry of drawable	XGetGeometry(3X)
XGetInputFocus: discover	current input focus window	XGetInputFocus(3X)
/obtain list of	current keyboard control	XGetKeyboardControl(3X)
XGetPointerControl: get	current pointer acceleration/	XGetPointerControl(3X)
XQueryPointer: get	current pointer location	XQueryPointer(3X)
XGetScreenSaver: get	current screen saver parameters	XGetScreenSaver(3X)
/obtain bit vector for	current state of keyboard	XQueryKeymap(3X)
XFreeCursor: destroy a	cursor	XFreeCursor(3X)
XRecolorCursor: change color of	cursor	XRecolorCursor(3X)
create a cursor from standard	cursor font XCreateFontCursor:	XCreateFontCursor(3X)
XCreateGlyphCursor: create a	cursor from font glyphs	XCreateGlyphCursor(3X)
XCreateFontCursor: create a	cursor from standard cursor font	XCreateFontCursor(3X)
XCreatePixmapCursor: create a	cursor from two bitmaps	XCreatePixmapCursor(3X)
XUndefineCursor: disassociate	cursor from window	XUndefineCursor(3X)
get closest supported	cursor sizes XQueryBestCursor:	XQueryBestCursor(3X)
/obtain the "best" supported	cursor, tile, or stipple size	XQueryBestSize(3X)
XDefineCursor: assign a	cursor to a window	XDefineCursor(3X)
X10) XDraw: draw polyline or	curve between vertex list (from	XDraw(3X)
/draw filled polygon or	curve from vertex list (from V10)	XDrawFilled(3X)
XFetchBytes: return data from	cut buffer 0	XFetchBytes(3X)
XStoreBytes: store data in	cut buffer 0	XStoreBytes(3X)
XFetchBuffer: return data from	cut buffer	XFetchBuffer(3X)
XStoreBuffer: store data in a	cut buffer	XStoreBuffer(3X)
XRotateBuffers: rotate the	cut buffers	XRotateBuffers(3X)
/set *dash_offset* and	*dash_list* (for lines) of/	XSetDashes(3X)
XSetDashes: set	*dash_offset* and/	XSetDashes(3X)
bitmap from X11 bitmap format	data /create	XCreateBitmapFromData(3X)
a pixmap with depth from bitmap	data /create	XCreatePixmapFromBitmapData(3
error messages from the error	data base /obtain	XGetErrorDatabaseText(3X)
XrmParseCommand: load resource	data base from command line/	XrmParseCommand(3X)
XFree: free specified in-memory	data created by an Xlib function	XFree(3X)
XLookUpAssoc: obtain	data from an association table	XLookUpAssoc(3X)
graphics/ XFindContext: get	data from context manager (not	XFindContext(3X)
XFetchBytes: return	data from cut buffer 0	XFetchBytes(3X)
XFetchBuffer: return	data from cut buffer	XFetchBuffer(3X)
XStoreBuffer: store	data in a cut buffer	XStoreBuffer(3X)
XStoreBytes: store	data in cut buffer 0	XStoreBytes(3X)
window and/ XSaveContext: save	data value corresponding to	XSaveContext(3X)
store a resource into a	database XrmPutResource:	XrmPutResource(3X)
store a resource into a	database XrmQPutResource:	XrmQPutResource(3X)
supported RGB/ XLookupColor: get	database and closest hardware	XLookupColor(3X)

XrmPutLineResource: add resource	entry given as string of name and/	XrmPutLineResource(3X)
XMakeAssoc: create	entry in an association table	XMakeAssoc(3X)
/set or change read/write	entry of colormap to closest/	XStoreColor(3X)
structure /add a new	entry to an **XModifierKeymap**	XInsertModifiermapEntry(3X)
obtain description of	error code XGetErrorText:	XGetErrorText(3X)
/obtain error messages from the	error data base	XGetErrorDatabaseText(3X)
XSetErrorHandler: set non-fatal	error event handler	XSetErrorHandler(3X)
XGetErrorDatabaseText: obtain	error messages from the error/	XGetErrorDatabaseText(3X)
handle fatal I/O	errors XSetIOErrorHandler:	XSetIOErrorHandler(3X)
/and wait for all events and	errors to be processed by server	XSync(3X)
check event queue for matching	event XCheckIfEvent:	XCheckIfEvent(3X)
XIfEvent: wait for matching	event	XIfEvent(3X)
XSendEvent: send an	event	XSendEvent(3X)
XPutBackEvent: push	event back on the input queue	XPutBackEvent(3X)
set non-fatal error	event handler XSetErrorHandler:	XSetErrorHandler(3X)
window /return next	event in queue matching type and	XCheckTypedWindowEvent(3X)
XCheckTypedEvent: return next	event in queue that matches event/	XCheckTypedEvent(3X)
XCheckWindowEvent: remove next	event matching passed window/	XCheckWindowEvent(3X)
XWindowEvent: remove next	event matching mask and window	XWindowEvent(3X)
XNextEvent: get next	event of any type or window	XNextEvent(3X)
check the number of events in the	event queue XEventsQueued:	XEventsQueued(3X)
XCheckIfEvent: check	event queue for matching event	XCheckIfEvent(3X)
not/ XCheckMaskEvent: remove next	event that matches mask but do	XCheckMaskEvent(3X)
XMaskEvent: remove next	event that matches passed mask	XMaskEvent(3X)
and/ XLookupString: map key	event to ASCII string, keysym,	XLookupString(3X)
/next event in queue that matches	event type but do not wait	XCheckTypedEvent(3X)
window XSelectInput: select the	event types to be sent to a	XSelectInput(3X)
the queue XPeekEvent: get	event without removing it from	XPeekEvent(3X)
the queue; do/ XPeekIfEvent: get	event without removing it from	XPeekIfEvent(3X)
get pointer motion	events XGetMotionEvents:	XGetMotionEvents(3X)
the number of pending input	events /output buffer and return	XPending(3X)
/output buffer and wait for all	events and errors to be processed/	XSync(3X)
/check the number of	events in the event queue	XEventsQueued(3X)
/behavior of keyboard and pointer	events when these resources are/	XAllowEvents(3X)
XNoOp: send a NoOp to	exercise connection with server	XNoOp(3X)
XShrinkRegion: reduce or	expand the size of a region	XShrinkRegion(3X)
XQueryExtension: get	extension information	XQueryExtension(3X)
allocated for list of installed	extensions to X /free memory	XFreeExtensionList(3X)
server /return list of all	extensions to X supported by the	XListExtensions(3X)
when connecting to that display	fails /reports the display name	XDisplayName(3X)
XSetIOErrorHandler: handle	fatal I/O errors	XSetIOErrorHandler(3X)
XWriteBitmapFile: write bitmap to	file	XWriteBitmapFile(3X)

retrieve a database from a	file XrmGetFileDatabase:	XrmGetFileDatabase(3X)
store database in	file XrmPutFileDatabase:	XrmPutFileDatabase(3X)
XGetDefault: scan user preference	file for program name and options	XGetDefault(3X)
XFillPolygon:	fill a polygon	XFillPolygon(3X)
XFillArc:	fill an arc	XFillArc(3X)
XLoadQueryFont: load a font and	fill information structure	XLoadQueryFont(3X)
XFillArcs:	fill multiple arcs	XFillArcs(3X)
XFillRectangles:	fill multiple rectangular areas	XFillRectangles(3X)
XFillRectangle:	fill rectangular area	XFillRectangle(3X)
XSetFillRule: set	fill rule in graphics context	XSetFillRule(3X)
XSetFillStyle: set	fill style in graphics context	XSetFillStyle(3X)
XSetTile: set	fill tile in graphics context	XSetTile(3X)
obtain the best supported	fill tile shape XQueryBestTile:	XQueryBestTile(3X)
vertex list/ XDrawFilled: draw	filled polygon or curve from	XDrawFilled(3X)
that matches/ XGetVisualInfo:	find visual information structure	XGetVisualInfo(3X)
XDrawArc: draws an arc	fitting inside a rectangle	XDrawArc(3X)
/set graphics-exposures	flag in graphics context	XSetGraphicsExposures(3X)
all events and errors to/ XSync:	flush output buffer and wait for	XSync(3X)
return the number of/ XPending:	flush the output buffer and	XPending(3X)
all queued requests) XFlush:	flush the output buffer (display	XFlush(3X)
discover current input	focus window XGetInputFocus:	XGetInputFocus(3X)
XSetInputFocus: set the input	focus window	XSetInputFocus(3X)
font if not already loaded; get	font ID XLoadFont: load	XLoadFont(3X)
a cursor from standard cursor	font XCreateFontCursor: create	XCreateFontCursor(3X)
return information about loaded	font XQueryFont:	XQueryFont(3X)
XUnloadFont: unload a	font	XUnloadFont(3X)
structure XLoadQueryFont: load a	font and fill information	XLoadQueryFont(3X)
	font displayer for X	xfd(1)
structure XFreeFont: unload	font and free storage for font	XFreeFont(3X)
create a cursor from	font glyphs XCreateGlyphCursor:	XCreateGlyphCursor(3X)
font ID XLoadFont: load	font if not already loaded; get	XLoadFont(3X)
XSetFont: set current	font in graphics context	XSetFont(3X)
XFreeFontInfo: free multiple	font information arrays	XFreeFontInfo(3X)
server	font list displayer for X	xlsfonts(1)
query server for string and	font metrics XQueryTextExtents:	XQueryTextExtents(3X)
XTextExtents: get string and	font metrics	XTextExtents(3X)
/query server for string and	font metrics of 16-bit character/	XQueryTextExtents16(3X)
XTextExtents16: get string and	font metrics of 16-bit character/	XTextExtents16(3X)
XFreeFontNames: free	font name array	XFreeFontNames(3X)
return a list of the available	font names XListFonts:	XListFonts(3X)
XGetFontProperty: get a	font property given its atom	XGetFontProperty(3X)
XGetFontPath: get the current	font search path	XGetFontPath(3X)

INDEX

XSetStandardProperties: set	minimum set of properties for/	XSetStandardProperties(3X)
XSetArcMode: set arc	mode in graphics context	XSetArcMode(3X)
SetSubwindowMode: set subwindow	mode in graphics context	XSetSubwindowMode(3X)
/change close down	mode of client	XSetCloseDownMode(3X)
XDisableAccessControl: prevent	modification to the host access/	XDisableAccessControl(3X)
/update the stored	modifier and keymap information	XRefreshKeyboardMapping(3X)
XGetModifierMapping: obtains	modifier key mapping (Shift,/	XGetModifierMapping(3X)
/create a keyboard	modifier mapping structure	XNewModifiermap(3X)
/destroy and free keyboard	modifier mapping table	XFreeModifiermap(3X)
set keycodes to be used as	modifiers XSetModifierMapping:	XSetModifierMapping(3X)
XGetMotionEvents: get pointer	motion events	XGetMotionEvents(3X)
XMoveWindow:	move a window	XMoveWindow(3X)
on screen XWarpPointer:	move the pointer to another point	XWarpPointer(3X)
XDrawArcs: draw	multiple arcs	XDrawArcs(3X)
XFillArcs: fill	multiple arcs	XFillArcs(3X)
XDrawLines: draw	multiple connected lines	XDrawLines(3X)
XDrawSegments: draw	multiple disjoint lines	XDrawSegments(3X)
XFreeFontInfo: free	multiple font information arrays	XFreeFontInfo(3X)
control/ XRemoveHosts: remove	multiple hosts from the access	XRemoveHosts(3X)
control list XAddHosts: add	multiple hosts to the access	XAddHosts(3X)
XDrawPoints: draw	multiple points	XDrawPoints(3X)
/draw the outlines of	multiple rectangles	XDrawRectangles(3X)
XFillRectangles: fill	multiple rectangular areas	XFillRectangles(3X)
XFetchName: get window	name (WM_NAME property)	XFetchName(3X)
read-only colorcell from color	name XAllocNamedColor: allocate	XAllocNamedColor(3X)
supported RGB values from color	name /and closest hardware	XLookupColor(3X)
read/write colorcell by color	name XStoreNamedColor: allocate	XStoreNamedColor(3X)
/get resource from	name and class as quarks	XrmQGetResource(3X)
XrmGetResource: get resource from	name and class as strings	XrmGetResource(3X)
user preference file for program	name and options /scan	XGetDefault(3X)
resource entry given as string of	name and value /add	XrmPutLineResource(3X)
XFreeFontNames: free font	name array	XFreeFontNames(3X)
XGetAtomName: get	name for atom	XGetAtomName(3X)
XInternAtom: return an atom for a	name string	XInternAtom(3X)
window's icon XSetIconName: set	name to be displayed in a	XSetIconName(3X)
XGetIconName: get	name to be displayed in icon	XGetIconName(3X)
XStringToKeysym: convert keysym	name to keysym code	XStringToKeysym(3X)
XStoreName: assign	name to window for manager	XStoreName(3X)
XDisplayName: reports the display	name when connecting to that/	XDisplayName(3X)
a list of the available font	names XListFonts: return	XListFonts(3X)
XListFontsWithInfo: obtain the	names and information about/	XListFontsWithInfo(3X)
XCheckTypedWindowEvent: return	next event in queue matching type/	XCheckTypedWindowEvent(3X)

/scan user preference file for	program name and options	XGetDefault(3X)
XOpenDisplay: connect a client	program to an X server	XOpenDisplay(3X)
/rotate properties in the	properties array	XRotateWindowProperties(3X)
/set minimum set of	properties for window manager	XSetStandardProperties(3X)
XRotateWindowProperties: rotate	properties in the properties/	XRotateWindowProperties(3X)
XDeleteProperty: delete a window	property	XDeleteProperty(3X)
get window name (WM_NAME	property) XFetchName:	XFetchName(3X)
set value of the WM_ICON_SIZE	property XSetIconSizes:	XSetIconSizes(3X)
set the window manager hints	property XSetWMHints:	XSetWMHints(3X)
XChangeProperty: change a	property associated with a window	XChangeProperty(3X)
	property displayer for X	xprop(1)
/set WM_TRANSIENT_FOR	property for window	XSetTransientForHint(3X)
XSetZoomHints: set size hints	property for zoomed windows	XSetZoomHints(3X)
/obtain the atom type and	property format for a window	XGetWindowProperty(3X)
XGetFontProperty: get a font	property given its atom	XGetFontProperty(3X)
XListProperties: get	property list for window	XListProperties(3X)
XGetClassHint: get the WM_CLASS	property of a window	XGetClassHint(3X)
/set the value of any	property of type SIZE_HINTS	XSetSizeHints(3X)
XGetSizeHints: read any	property of type/	XGetSizeHints(3X)
/get WM_TRANSIENT_FOR	property of window	XGetTransientForHint(3X)
XSetClassHint: set WM_CLASS	property of window	XSetClassHint(3X)
queue XPutBackEvent:	push event back on the input	XPutBackEvent(3X)
convert a string to a	quark XrmStringToQuark:	XrmStringToQuark(3X)
XrmUniqueQuark: allocate a new	quark	XrmUniqueQuark(3X)
key string to binding list and	quark list /convert	XrmStringToBindingQuarkList(3X)
convert key string to	quark list XrmStringToQuarkList:	XrmStringToQuarkList(3X)
XrmQuarkToString: convert a	quark to a string	XrmQuarkToString(3X)
resource from name and class as	quarks XrmQGetResource: get	XrmQGetResource(3X)
resource value to database using	quarks /add a string	XrmQPutStringResource(3X)
metrics XQueryTextExtents:	query server for string and font	XQueryTextExtents(3X)
metrics of/ XQueryTextExtents16:	query server for string and font	XQueryTextExtents16(3X)
the number of events in the event	queue XEventsQueued: check	XEventsQueued(3X)
without removing it from the	queue XPeekEvent: get event	XPeekEvent(3X)
push event back on the input	queue XPutBackEvent:	XPutBackEvent(3X)
without removing it from the	queue; do not wait /get event	XPeekIfEvent(3X)
XCheckIfEvent: check event	queue for matching event	XCheckIfEvent(3X)
/return next event in	queue matching type and window	XCheckTypedWindowEvent(3X)
do not wait /return next event in	queue that matches event type but	XCheckTypedEvent(3X)
the output buffer (display all	queued requests) XFlush: flush	XFlush(3X)
order XRaiseWindow:	raise a window to top of stacking	XRaiseWindow(3X)
XReadBitmapFile:	read a bitmap from disk	XReadBitmapFile(3X)
XGetSizeHints:	read any property of type/	XGetSizeHints(3X)

XUngrabKeyboard:	release keyboard from grab	XUngrabKeyboard(3X)
XUngrabServer:	release server from grab	XUngrabServer(3X)
XUngrabPointer:	release the pointer from grab	XUngrabPointer(3X)
client's/ XChangeSaveSet: add or	remove a subwindow from the	XChangeSaveSet(3X)
list XRemoveHost:	remove host from access control	XRemoveHost(3X)
access control/ XRemoveHosts:	remove multiple hosts from the	XRemoveHosts(3X)
assed window/ XCheckWindowEvent:	remove next event matching both	XCheckWindowEvent(3X)
and window XWindowEvent:	remove next event matching mask	XWindowEvent(3X)
mask but do not/ XCheckMaskEvent:	remove next event that matches	XCheckMaskEvent(3X)
passed mask XMaskEvent:	remove next event that matches	XMaskEvent(3X)
the client's/ XRemoveFromSaveSet:	removes a window's children from	XRemoveFromSaveSet(3X)
XPeekEvent: get event without	removing it from the queue	XPeekEvent(3X)
XPeekIfEvent: get event without	removing it from the queue; do/	XPeekIfEvent(3X)
connecting to that/ XDisplayName:	reports the display name when	XDisplayName(3X)
output buffer (display all queued	requests) XFlush: flush the	XFlush(3X)
XResetScreenSaver:	reset the screen saver	XResetScreenSaver(3X)
determine if rectangle	resides in region XRectInRegion:	XRectInRegion(3X)
database levels for a given	resource /search resource	XrmQGetSearchResource(3X)
line/ XrmParseCommand: load	resource data base from command	XrmParseCommand(3X)
XrmQGetSearchResource: search	resource database levels for a/	XrmQGetSearchResource(3X)
name and/ XrmPutLineResource: add	resource entry given as string of	XrmPutLineResource(3X)
quarks XrmQGetResource: get	resource from name and class as	XrmQGetResource(3X)
strings XrmGetResource: get	resource from name and class as	XrmGetResource(3X)
XrmPutResource: store a	resource into a database	XrmPutResource(3X)
XrmQPutResource: store a	resource into a database	XrmQPutResource(3X)
XrmInitialize: initialize the	resource manager	XrmInitialize(3X)
XrmPutStringResource: add a	resource that is specified as a/	XrmPutStringResource(3X)
quarks /add a string	resource value to database using	XrmQPutStringResource(3X)
destroy a client or its remaining	resources XKillClient:	XKillClient(3X)
and pointer events when these	resources are grabbed /keyboard	XAllowEvents(3X)
XrmGetFileDatabase:	retrieve a database from a file	XrmGetFileDatabase(3X)
XrmQGetSearchList:	return a list of database levels	XrmQGetSearchList(3X)
font names XListFonts:	return a list of the available	XListFonts(3X)
XInternAtom:	return an atom for a name string	XInternAtom(3X)
XFetchBytes:	return data from cut buffer 0	XFetchBytes(3X)
XFetchBuffer:	return data from cut buffer	XFetchBuffer(3X)
font XQueryFont:	return information about loaded	XQueryFont(3X)
X supported by/ XListExtensions:	return list of all extensions to	XListExtensions(3X)
/copy a colormap and	return new colormap ID	XCopyColormapAndFree(3X)
ıatching/ XCheckTypedWindowEvent:	return next event in queue	XCheckTypedWindowEvent(3X)
matches event/ XCheckTypedEvent:	return next event in queue that	XCheckTypedEvent(3X)
XGetSelectionOwner:	return selection owner	XGetSelectionOwner(3X)

XGetKeyboardMapping:	return symbols for keycodes	XGetKeyboardMapping(3X)
/flush the output buffer and	return the number of pending/	XPending(3X)
XBell:	ring the bell (Control G)	XBell(3X)
a list of children, parent, and	root XQueryTree: obtains	XQueryTree(3X)
utility for X	root window parameter setting	xsetroot(1)
XRotateWindowProperties:	rotate properties in the/	XRotateWindowProperties(3X)
XRotateBuffers:	rotate the cut buffers	XRotateBuffers(3X)
XSetFillRule: set fill	rule in graphics context	XSetFillRule(3X)
window and context/ XSaveContext:	save data value corresponding to	XSaveContext(3X)
reset the screen	saver XResetScreenSaver:	XResetScreenSaver(3X)
set parameters of the screen	saver XSetScreenSaver:	XSetScreenSaver(3X)
XForceScreenSaver: turn screen	saver on or off	XForceScreenSaver(3X)
/get current screen	saver parameters	XGetScreenSaver(3X)
add window's children to client's	save-set XAddToSaveSet:	XAddToSaveSet(3X)
a subwindow from the client's	save-set /add or remove	XChangeSaveSet(3X)
children from the client's	save-set /removes a window's	XRemoveFromSaveSet(3X)
program name and/ XGetDefault:	scan user preference file for	XGetDefault(3X)
the pointer to another point on	screen XWarpPointer: move	XWarpPointer(3X)
XActivateScreenSaver: activate	screen blanking	XActivateScreenSaver(3X)
XResetScreenSaver: reset the	screen saver	XResetScreenSaver(3X)
set parameters of the	screen saver XSetScreenSaver:	XSetScreenSaver(3X)
XForceScreenSaver: turn	screen saver on or off	XForceScreenSaver(3X)
XGetScreenSaver: get current	screen saver parameters	XGetScreenSaver(3X)
get the current font	search path XGetFontPath:	XGetFontPath(3X)
XSetFontPath: set the font	search path	XSetFontPath(3X)
for a/ XrmQGetSearchResource:	search resource database levels	XrmQGetSearchResource(3X)
to a window XSelectInput:	select the event types to be sent	XSelectInput(3X)
use the value of a	selection XConvertSelection:	XConvertSelection(3X)
set owner of a	selection XSetSelectionOwner:	XSetSelectionOwner(3X)
XGetSelectionOwner: return	selection owner	XGetSelectionOwner(3X)
connection with server XNoOp:	send a NoOp to exercise	XNoOp(3X)
XSendEvent:	send an event	XSendEvent(3X)
select the event types to be	sent to a window XSelectInput:	XSelectInput(3X)
XGrabServer: grab the	server	XGrabServer(3X)
extensions to X supported by the	server /return list of all	XListExtensions(3X)
NoOp to exercise connection with	server XNoOp: send a	XNoOp(3X)
connect a client program to an X	server XOpenDisplay:	XOpenDisplay(3X)
and errors to be processed by	server /and wait for all events	XSync(3X)
/disconnect client from an X	server and display	XCloseDisplay(3X)
X	server font list displayer for	xlsfonts(1)
metrics XQueryTextExtents: query	server for string and font	XQueryTextExtents(3X)
XQueryTextExtents16: query	server for string and font/	XQueryTextExtents16(3X)

XUngrabServer: release	server from grab	XUngrabServer(3X)
XSetClassHint:	set WM_CLASS property of window ...	XSetClassHint(3X)
for window XSetTransientForHint:	set WM_TRANSIENT_FOR property ...	XSetTransientForHint(3X)
XPutPixel:	set a pixel value in an image	XPutPixel(3X)
XSetArcMode:	set arc mode in graphics context	XSetArcMode(3X)
create a window and	set attributes XCreateWindow:	XCreateWindow(3X)
graphics context XSetBackground:	set background pixel value in	XSetBackground(3X)
graphics context XSetFunction:	set bitwise logical operation in	XSetFunction(3X)
context XSetClipOrigin:	set clip origin in graphics	XSetClipOrigin(3X)
context XSetClipMask:	set clip_mask pixmap in graphics	XSetClipMask(3X)
context XSetFont:	set current font in graphics	XSetFont(3X)
dash_list/ XSetDashes:	set *dash_offset* and	XSetDashes(3X)
XSetFillRule:	set fill rule in graphics context	XSetFillRule(3X)
context XSetFillStyle:	set fill style in graphics	XSetFillStyle(3X)
XSetTile:	set fill tile in graphics context	XSetTile(3X)
logical function and/ XSetState:	set foreground, background,	XSetState(3X)
graphics context XSetForeground:	set foreground pixel value in	XSetForeground(3X)
XSetAfterFunction:	set function called after all Xli	XSetAfterFunction(3X)
graphics/ XSetGraphicsExposures:	set graphics-exposures flag in	XSetGraphicsExposures(3X)
modifiers XSetModifierMapping:	set keycodes to be used as	XSetModifierMapping(3X)
graphics/ XSetLineAttributes:	set line drawing components in	XSetLineAttributes(3X)
window/ XSetStandardProperties:	set minimum set of properties for	XSetStandardProperties(3X)
window's icon XSetIconName:	set name to be displayed in a	XSetIconName(3X)
XSetErrorHandler:	set non-fatal error event handler	XSetErrorHandler(3X)
manager /set minimum	set of properties for window	XSetStandardProperties(3X)
colormap to closest/ XStoreColor:	set or change read/write entry of	XStoreColor(3X)
XSetSelectionOwner:	set owner of a selection	XSetSelectionOwner(3X)
saver XSetScreenSaver:	set parameters of the screen	XSetScreenSaver(3X)
context XSetPlaneMask:	set plane mask in graphics	XSetPlaneMask(3X)
XSetPointerMapping:	set pointer button mapping	XSetPointerMapping(3X)
normal state/ XSetNormalHints:	set size hints for window in	XSetNormalHints(3X)
zoomed windows XSetZoomHints:	set size hints property for	XSetZoomHints(3X)
XSetStipple:	set stipple in graphics context	XSetStipple(3X)
context XSetSubwindowMode:	set subwindow mode in graphics	XSetSubwindowMode(3X)
(command line args) XSetCommand:	set the WM_COMMAND atom	XSetCommand(3X)
attribute/ XSetWindowBackground:	set the background pixel	XSetWindowBackground(3X)
context to the/ XSetRegion:	set the clip_mask of the graphics	XSetRegion(3X)
window XSetWindowColormap:	set the colormap for a specified	XSetWindowColormap(3X)
XSetFontPath:	set the font search path	XSetFontPath(3X)
XSetInputFocus:	set the input focus window	XSetInputFocus(3X)
type SIZE_HINTS XSetSizeHints:	set the value of any property of	XSetSizeHints(3X)
property XSetWMHints:	set the window manager hints	XSetWMHints(3X)

INDEX

/find visual information	structure that matches template	XGetVisualInfo(3X)
XSetFillStyle: set fill	style in graphics context	XSetFillStyle(3X)
XSubImage: create	subimage from part of image	XSubImage(3X)
XSubtractRegion:	subtract one region from another	XSubtractRegion(3X)
XChangeSaveSet: add or remove a	subwindow from the client's/	XChangeSaveSet(3X)
context XSetSubwindowMode: set	subwindow mode in graphics	XSetSubwindowMode(3X)
and destroy a window and all	subwindows /unmap	XDestroyWindow(3X)
XMapSubwindows: map all	subwindows	XMapSubwindows(3X)
XUnmapSubwindows: unmap all	subwindows of a given window	XUnmapSubwindows(3X)
XDestroySubwindows: destroy all	subwindows of a window	XDestroySubwindows(3X)
/get database and closest hardware	supported RGB values from color/	XLookupColor(3X)
list of all extensions to X	supported by the server /return	XListExtensions(3X)
XQueryBestCursor: get closest	supported cursor sizes	XQueryBestCursor(3X)
stipple size /obtain the "best"	supported cursor, tile, or	XQueryBestSize(3X)
XQueryBestTile: obtain the best	supported fill tile shape	XQueryBestTile(3X)
/obtain the best	supported stipple shape	XQueryBestStipple(3X)
convert *KeySym*	symbol to ASCII XKeysymToString:	XKeysymToString(3X)
XGetKeyboardMapping: return	symbols for keycodes	XGetKeyboardMapping(3X)
XSynchronize: enable or disable	synchronization for debugging	XSynchronize(3X)
/change coordinate	system from one window to another ...	XTranslateCoordinates(3X)
create a new association	table (X10) XCreateAssocTable:	XCreateAssocTable(3X)
an entry from an association	table XDeleteAssoc: delete	XDeleteAssoc(3X)
memory allocated for association	table /free the	XDestroyAssocTable(3X)
free keyboard modifier mapping	table /destroy and	XFreeModifiermap(3X)
obtain data from an association	table XLookUpAssoc:	XLookUpAssoc(3X)
create entry in an association	table XMakeAssoc:	XMakeAssoc(3X)
structure that matches	template /find visual information	XGetVisualInfo(3X)
	terminal emulator for X	xterm(1)
/draw 16-bit image	text characters	XDrawImageString16(3X)
/draw 8-bit image	text characters	XDrawImageString(3X)
XDrawString: draw 8-bit	text string, foreground only	XDrawString(3X)
XDrawString16: draw two-byte	text strings	XDrawString16(3X)
/change window border	tile attribute	XSetWindowBorderPixmap(3X)
/change background	tile attribute of a window	XSetWindowBackgroundPixmap(
XSetTile: set fill	tile in graphics context	XSetTile(3X)
the "best" supported cursor,	tile, or stipple size /obtain	XQueryBestSize(3X)
obtain the best supported fill	tile shape XQueryBestTile:	XQueryBestTile(3X)
context XSetTSOrigin: set	tile/stipple origin in graphics	XSetTSOrigin(3X)
XCirculateSubwindowsUp: circulate	top child to bottom of stacking/	XCirculateSubwindowsUp(3X)
XMapRaised: map a window on	top of its siblings	XMapRaised(3X)
/circulate bottom child to	top of stacking order	XCirculateSubwindowsDown(3X)
XRaiseWindow: raise a window to	top of stacking order	XRaiseWindow(3X)

INDEX

color XParseColor: lookup or	translate RGB values from ASCII	XParseColor(3X)
keys XAutoRepeatOff:	turn off the keyboard auto-repeat	XAutoRepeatOff(3X)
keys XAutoRepeatOn:	turn on the keyboard auto-repeat	XAutoRepeatOn(3X)
XForceScreenSaver:	turn screen saver on or off	XForceScreenSaver(3X)
XDrawString16: draw	two-byte text strings	XDrawString16(3X)
set the value of any property of	type SIZE_HINTS XSetSizeHints:	XSetSizeHints(3X)
entry for given window and	type /delete context	XDeleteContext(3X)
window /obtain the atom	type and property format for a	XGetWindowProperty(3X)
next event in queue matching	type and window /return	XCheckTypedWindowEvent(3X)
event in queue that matches event	type but do not wait /return next	XCheckTypedEvent(3X)
/read any property of	type *WM_SIZE_HINTS*	XGetSizeHints(3X)
/to window and context	type (not graphics context)	XSaveContext(3X)
XNextEvent: get next event of any	type or window	XNextEvent(3X)
default if/ XUninstallColormap:	uninstall colormap, install	XUninstallColormap(3X)
/the difference between the	union and intersection of two/	XXorRegion(3X)
XUnionRegion: compute the	union of two regions	XUnionRegion(3X)
XUnloadFont:	unload a font	XUnloadFont(3X)
font structure XFreeFont:	unload font and free storage for	XFreeFont(3X)
window XUnmapSubwindows:	unmap all subwindows of a given	XUnmapSubwindows(3X)
all subwindows XDestroyWindow:	unmap and destroy a window and	XDestroyWindow(3X)
XCreateSimpleWindow: creates an	unmapped *InputOutput* window	XCreateSimpleWindow(3X)
XUnmapWindow:	unmaps a window	XUnmapWindow(3X)
eymap/ XRefreshKeyboardMapping:	update the stored modifier and	XRefreshKeyboardMapping(3X)
XConvertSelection:	use the value of a selection	XConvertSelection(3X)
/calculate window geometry given	user geometry string and default/	XGeometry(3X)
name and/ XGetDefault: scan	user preference file for program	XGetDefault(3X)
	user preference utility for X	xset(1)
window border attribute to pixel	value XSetWindowBorder: change	XSetWindowBorder(3X)
values for the specified pixel	value /obtains the RGB	XQueryColor(3X)
entry given as string of name and	value /add resource	XrmPutLineResource(3X)
context/ XSaveContext: save data	value corresponding to window and	XSaveContext(3X)
XGetPixel: obtain a single pixel	value from an image	XGetPixel(3X)
XPutPixel: set a pixel	value in an image	XPutPixel(3X)
/set background pixel	value in graphics context	XSetBackground(3X)
/set foreground pixel	value in graphics context	XSetForeground(3X)
add constant value to every pixel	value in image XAddPixel:	XAddPixel(3X)
XConvertSelection: use the	value of a selection	XConvertSelection(3X)
XSetSizeHints: set the	value of any property of type/	XSetSizeHints(3X)
property XSetIconSizes: set	value of the WM_ICON_SIZE	XSetIconSizes(3X)
/add a string resource	value to database using quarks	XrmQPutStringResource(3X)
image XAddPixel: add constant	value to every pixel value in	XAddPixel(3X)
RGB values for an array of pixel	values XQueryColors: obtain	XQueryColors(3X)

values XQueryColors: obtain RGB	values for an array of pixel	XQueryColors(3X)
XQueryColor: obtains the RGB	values for the specified pixel/	XQueryColor(3X)
/lookup or translate RGB	values from ASCII color	XParseColor(3X)
closest hardware supported RGB	values from color name /and	XLookupColor(3X)
XQueryKeymap: obtain bit	vector for current state of/	XQueryKeymap(3X)
/draw filled polygon or curve from	vertex list (from V10)	XDrawFilled(3X)
draw polyline or curve between	vertex list (from X10) XDraw:	XDraw(3X)
matches/ XGetVisualInfo: find	visual information structure that	XGetVisualInfo(3X)
the/ XMatchVisualInfo: obtain the	visual information that matches	XMatchVisualInfo(3X)
that matches mask but do not	wait /remove next event	XCheckMaskEvent(3X)
matches event type but do not	wait /next event in queue that	XCheckTypedEvent(3X)
and passed mask, but do not	wait /matching both passed window	XCheckWindowEvent(3X)
it from the queue; do not	wait /get event without removing	XPeekIfEvent(3X)
XSync: flush output buffer and	wait for all events and errors to/	XSync(3X)
XIfEvent:	wait for matching event	XIfEvent(3X)
character/ XTextWidth16: get	width in pixels of 16-bit	XTextWidth16(3X)
character string XTextWidth: get	width in pixels of 8-bit	XTextWidth(3X)
/change the border	width of a window	XSetWindowBorderWidth(3X)
window position, size, border	width, or stacking order /change	XConfigureWindow(3X)
a property associated with a	window XChangeProperty: change	XChangeProperty(3X)
event in queue matching type and	window /return next	XCheckTypedWindowEvent(3
clear a rectangular area in a	window XClearArea:	XClearArea(3X)
XClearWindow: clear an entire	window	XClearWindow(3X)
an unmapped *InputOutput*	window /creates	XCreateSimpleWindow(3X)
assign a cursor to a	window XDefineCursor:	XDefineCursor(3X)
destroy all subwindows of a	window XDestroySubwindows:	XDestroySubwindows(3X)
get the WM_CLASS property of a	window XGetClassHint:	XGetClassHint(3X)
discover current input focus	window XGetInputFocus:	XGetInputFocus(3X)
get WM_TRANSIENT_FOR property of	window XGetTransientForHint:	XGetTransientForHint(3X)
obtain current attributes of	window XGetWindowAttributes:	XGetWindowAttributes(3X)
type and property format for a	window /obtain the atom	XGetWindowProperty(3X)
read size hints for zoomed	window XGetZoomHints:	XGetZoomHints(3X)
get property list for	window XListProperties:	XListProperties(3X)
XMapWindow: map a	window	XMapWindow(3X)
change size and location of	window XMoveResizeWindow:	XMoveResizeWindow(3X)
XMoveWindow: move a	window	XMoveWindow(3X)
get next event of any type or	window XNextEvent:	XNextEvent(3X)
the event types to be sent to a	window XSelectInput: select	XSelectInput(3X)
set WM_CLASS property of	window XSetClassHint:	XSetClassHint(3X)
set the input focus	window XSetInputFocus:	XSetInputFocus(3X)
set WM_TRANSIENT_FOR property for	window XSetTransientForHint:	XSetTransientForHint(3X)
background pixel attribute of a	window /set the	XSetWindowBackground(3X)

MOTOROLA INC.

Microcomputer Division
2900 South Diablo Way
Tempe, Arizona 85282
P.O. Box 2953
Phoenix, Arizona 85062

10374 PRINTED IN USA 11-88 IRONWOOD LITHO 8238-8 1,500