The Definitive Guides to the X Window System

Volume Two

Xlib Reference Manual

for Version 11

O'Reilly & Associates, Inc.

Volume Two

Xlib Reference Manual

for Version 11 of the X Window System

edited by Adrian Nye

O'Reilly & Associates, Inc.

Revision and Printing History

August 1988:	First Printing.
November 1988:	Second Printing. Minor revisions.
May 1989:	Third Printing. Release 3 updates added. Minor revisions.
April 1990:	Second Edition covers Release 3 and Release 4. Major revisions.
July 1990:	Fifth Printing. Minor revisions.

Small Print

Volume 2: ISBN 0-937175-12-9 Set: ISBN 0-937175-13-7

Table of Contents

Preface

About This Manual

This manual describes the X library, the C Language programming interface to Version 11 of the X Window System. The X library, known as Xlib, is the lowest level of programming interface to X. This library enables a programmer to write applications with an advanced user interface based on windows on the screen, with complete network transparency, that will run without changes on many types of workstations and personal computers.

Xlib is powerful enough to write effective applications without additional programming tools and is necessary for certain tasks even in applications written with higher-level "toolkits."

There are a number of these toolkits for X programming, the most notable being the DEC/MIT toolkit Xt, the Andrew toolkit developed by IBM and Carnegie-Mellon University, and the InterViews toolkit from Stanford. These toolkits are still evolving, and only Xt is currently part of the X standard. Toolkits simplify the process of application writing considerably, providing a number of *widgets* that implement menus, command buttons, and other common features of the user interface.

This manual does not describe Xt or any other toolkit. That is done in Volumes Four, Five, and Six of our X Window System series. Nonetheless, much of the material described in this book is helpful for understanding and using the toolkits, since the toolkits themselves are written using Xlib and allow Xlib code to be intermingled with toolkit code.

Summary of Contents

This manual is divided into two volumes. This is the second volume, the *Xlib Reference Manual*. It includes reference pages for each of the Xlib functions (organized alphabetically), a permuted index, and numerous appendices and quick reference aids.

The first volume, the *Xlib Programming Manual*, provides a conceptual introduction to Xlib, including tutorial material and numerous programming examples. Arranged by task or topic, each chapter brings together a group of Xlib functions, describes the conceptual foundation they are based on, and illustrates how they are most often used in writing applications (or, in the case of the last chapter, in writing window managers). Volume One is structured so as to be useful as a tutorial and also as a task-oriented reference.

Volume One and Volume Two are designed to be used together. To get the most out of the examples in Volume One, you will need the exact calling sequences of each function from Volume Two. To understand fully how to use each of the functions described in Volume Two, all but the most experienced X "hacker" will need the explanation and examples in Volume One.

Both volumes include material from the original Xlib and X11 Protocol documentation provided by MIT, as well as from other documents provided on the MIT release tape. We have done our best to incorporate all of the useful information from the MIT documentation, to correct references we found to be in error, to reorganize and present it in a more useful form, and to supplement it with conceptual material, tutorials, reference aids, and examples. In other words, this manual is not only a replacement but is a superset of the MIT documentation.

Those of you familiar with the MIT documentation will recognize that each reference page in Volume Two includes the detailed description of the routine found in Gettys, Newman, and Scheifler's *Xlib–C Language X Interface*, plus, in many cases, additional text that clarifies ambiguities and describes the context in which the routine would be used. We have also added cross references to related reference pages and to where additional information can be found in Volume One.

How to Use This Manual

Volume Two is designed to make it as easy and fast as possible to look up virtually any fact about Xlib. It includes a permuted index, reference pages for each library function, appendices that cover macros, structures, function groups, events, fonts, colors, cursors, keysyms, and errors, and at-a-glance tables for the graphics context and window attributes.

The permuted index is the standard UNIX way of finding a particular function name given a keyword. By looking up a word in the second column that you think describes the function you are looking for, you can find the group of functions that have that word in their description lines. The description line also appears at the top of each reference page. Once you have found the routine you are looking for, you can look for its reference page.

The reference pages themselves provide all the details necessary for calling each routine, including its arguments, returned values, definitions of the structure types of arguments and returned values, and the errors it may generate. Many of the pages also give hints about how the routine is used in the context of other routines. This is the part of this volume you will use the most.

Appendix A, *Function Group Summary*, groups the routines according to function, and provides brief descriptions. You'll find it useful to have in one place a description of related routines, so their differences can be noted and the appropriate one chosen.

Appendix B, *Error Messages and Protocol Requests*, describes the errors that Xlib routines can generate. When an error is handled by the default error handler, one of these messages is printed. Also printed is the X Protocol request that caused the error. Since Protocol requests do not map directly to Xlib routines, this appendix provides a table with which you can find out which Xlib routine in your code caused the error.

Appendix C, *Macros*, describes the macros that access members of the `Display` structure, classify keysyms, and convert resource manager types.

Appendix D, *ColorCaEE*, presents the standard color database. The color names in this database should be available on all servers, though the corresponding RGB values may have been modified to account for screen variations.

Appendix E, *Event Reference*, describes each event type and structure, in a reference page format. This is an invaluable reference for event programming.

Appendix F, *Structure Reference*, describes all structures used by Xlib except the event structures described in Appendix E, including which routines use each structure.

Appendix G, *Symbol Reference*, lists in alphabetical order and describes all of the symbols defined in Xlib include files.

Appendix H, *Keysym Reference*, lists and describes each character in the standard keysym families, used for translating keyboard events. The characters for English and foreign language keysyms are shown where possible.

Appendix I, *The Cursor Font*, describes the standard cursor font, including a illustration of the font shapes.

Appendix J, *The Xmu Library*, provides reference pages for each function in the miscellaneous utilities library. This library is provided with the standard X distribution and is very useful when programming with Xlib.

Finally, Volume Two concludes with at-a-glance charts that help in setting the graphics context (GC) and the window attributes.

Example Programs

The example programs in this book are on the X11 Release 4 distribution in the contributed section. There are many ways of getting this distribution; most are described in Appendix H.

The example programs are also available free from UUNET (that is, free except for UUNET's usual connect-time charges). If you have access to UUNET, you can retrieve the source code using *uucp* or *ftp*. For *uucp*, find a machine with direct access to UUNET and type the following command:

```
    uucp uunet\!~uucp/nutshell/Xlib/xlibprgs.tar.Z yourhost\!~/yourname/
```

The backslashes can be omitted if you use the Bourne shell (*sh*) instead of *csh*. The file should appear some time later (up to a day or more) in the directory */usr/spool/uucppublic/**yourname***.

To use *ftp*, *ftp* to *uunet.uu.net* and use *anonymous* as your user name and *guest* as your password. Then type the following:

```
    cd /nutshell/Xlib
    binary (you must specify binary transfer for compressed files)
    get xlibprgs.tar.Z
    bye
```

The file is a compressed tar archive. To restore the files once you have retrieved the archive, type:

```
uncompress xlibprgs.tar
tar xvf xlibprgs.tar
```

The example programs are also available free by *ftp* from *expo.lcs.mit.edu*. The directory containing the examples is *contrib/examples/OReilly/Xlib*.

The examples will be installed in subdirectories under the current directory, one for each chapter in the book. Imakefiles are included. (Imakefiles are used with *imake*, a program supplied with the X11 distribution that generates proper Makefiles on a wide variety of systems.)

Assumptions

Readers should be 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 will be helpful.

Font Conventions Used in This Manual

Italic is used for:

- UNIX pathnames, filenames, program names, user command names, and options for user commands.
- New terms where they are defined.

`Typewriter Font` is used for:

- Anything that would be typed verbatim into code, such as examples of source code and text on the screen.
- The contents of include files, such as structure types, structure members, symbols (defined constants and bit flags), and macros.
- Xlib functions.
- Names of subroutines of the example programs.

`Italic Typewriter Font` is used for:

- Arguments to Xlib functions, since they could be typed in code as shown but are arbitrary.

Helvetica Italics are used for:

- Titles of examples, figures, and tables.

Boldface is used for:

- Chapter and section headings.

Related Documents

The C Programming Language by B. W. Kernighan and D. M. Ritchie

The following documents are included on the X11 source tape:

Xt Toolkit Intrinsics by Joel McCormack, Paul Asente, and Ralph Swick
Xt Toolkit Widgets by Ralph Swick and Terry Weissman
Xlib–C Language X Interface by Jim Gettys, Ron Newman, and Robert Scheifler
X Window System Protocol, Version 11 by Robert Scheifler

The following books on the X Window System are available from O'Reilly and Associates, Inc.:

Volume Zero — *X Protocol Reference Manual*
Volume Three — *X Window System User's Guide*
Volume Four — *X Toolkit Intrinsics Programming Manual*
Volume Five — *X Toolkit Intrinsics Reference Manual*
Volume Six — *X Toolkit Widgets Reference Manual* (available summer 1990)
Volume Seven — *XView Programmer's Guide*
Quick Reference — *The X Window System in a Nutshell*

Requests for Comments

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

Our U.S. mail address, e-mail address, and telephone number are as follows:

O'Reilly and Associates, Inc.
632 Petaluma Avenue
Sebastopol, CA 95472
(800) 338-6887

UUCP: uunet!ora!adrian ARPA: adrian@ora.UU.NET

Bulk Sales Information

This manual is being resold as the official X Window System documentation by many workstation manufacturers. For information on volume discounts for bulk purchase, call Linda Walsh at O'Reilly and Associates, Inc., at 617-354-5800, or send e-mail to linda@ora.com.

For companies requiring extensive customization of the book, source licensing terms are also available.

Acknowledgements

The information contained in this manual is based in part on *Xlib–C Language X Interface*, written by Jim Gettys, Ron Newman, and Robert Scheifler, and the *X Window System Protocol, Version 11*, by Robert Scheifler (with many contributors). The X Window System software and these documents were written under the auspices of Project Athena at MIT. In addition, this manual includes material from Oliver Jones' Xlib tutorial presentation, which was given at the MIT X Conference in January 1988, and from David Rosenthal's *Inter-Client Communication Conventions Manual*.

I would like to thank the people who helped this book come into being. It was Tim O'Reilly who originally sent me out on a contract to write a manual for X Version 10 for a workstation manufacturer and later to another company to write a manual for X Version 11, from which this book began. I have learned most of what I know about computers and technical writing while working for Tim. For this book, he acted as an editor, he helped me reorganize several chapters, he worked on the *Color* and *Managing User Preferences* chapters when time was too short for me to do it, and he kept my spirits up through this long project. While I was concentrating on the details, his eye was on the overall presentation, and his efforts improved the book enormously.

This book would not be as good (and we might still be working on it) had it not been for Daniel Gilly. Daniel was my production assistant for critical periods in the project. He dealt with formatting issues, checked for consistent usage of terms and noticed irregularities in content, and edited files from written corrections by me and by others. His job was to take as much of the work off me as possible, and with his technical skill and knowledge of UNIX, he did that very well.

This manual has benefitted from the work and assistance of the entire staff of O'Reilly and Associates, Inc. Susan Willing was responsible for graphics and design, and she proofed many drafts of the book; Linda Mui tailored the troff macros to the design by Sue Willing and myself and was invaluable in the final production process; John Strang figured out the resource manager and wrote the original section on that topic; Karen Cakebread edited a draft of the manual and established some conventions for terms and format. Peter Mui executed the "at-a-glance" tables for the inside back cover; Tom Scanlon entered written edits and performed copy fitting; Donna Woonteiler wrote the index of the book, Valerie Quercia, Tom Van Raalte, and Linda Walsh all contributed in some small ways; and Cathy Brennan, Suzanne Van Hove, and Jill Berlin fielded many calls from people interested in the X manual and saved me all the time that would have taken. Ruth Terry, Lenny Muellner, and Donna

Woonteiler produced the Second Edition, with graphics done by Chris Reilly. A special thanks to everyone at O'Reilly and Associates for putting up with my habits of printer and terminal hogging, lugging X books around, recycling paper, and for generally being good at what they do and good-natured to boot.

Many people sent in corrections for this Second Edition of the manual. Those whose efforts were most noteworthy were Jane-Na Chang of NEC, Jonathan Saunders of Identification and Security Systems Inc., Saundra Miller, and Russell Ferriday.

I would also like to thank the people from other companies that reviewed the book or otherwise made this project possible: John Posner, Barry Kingsbury, Jeff MacMann and Jeffrey Vroom of Stellar Computer; Oliver Jones of Apollo Computer; Sam Black, Jeff Graber, and Janet Egan of Masscomp; Al Tabayoyon, Paul Shearer, and many others from Tektronix; Robert Scheifler and Jim Fulton of the X Consortium (who helped with the *Color* and *Managing User Preferences* chapters), and Peter Winston II and Aub Harden of Integrated Computer Solutions. Despite the efforts of the reviewers and everyone else, any errors that remain are my own.

— *Adrian Nye*

Permuted Index

How to Use the Permuted Index

The permuted index takes the brief descriptive string from the title of each command page and rotates (permutes) the string so that each keyword will at one point start the *second*, or center, column of the line. The beginning and end of the original string are indicated by a slash when they are in other than their original position; if the string is too long, it is truncated.

To find the command you want, simply scan down the middle of the page, looking for a keyword of interest on the right side of the blank gutter. Once you find the keyword you want, you can read (with contortions) the brief description of the command that makes up the entry. If things still look promising, you can look all the way over to the right for the name of the relevant command page.

The Permuted Index

for string and font metrics of a	16-bit character string /server	XQueryTextExtents16
/get string and font metrics of a	16-bit character string, locally	XTextExtents16
/get the width in pixels of a	16-bit character string, locally	XTextWidth16
XDrawImageString16: draw	16-bit image text characters	XDrawImageString16
XDrawText16: draw	16-bit polytext strings	XDrawText16
/get the width in pixels of an	8-bit character string, locally	XTextWidth
XDrawImageString: draw	8-bit image text characters	XDrawImageString
XDrawText: draw	8-bit polytext strings	XDrawText
only XDrawString: draw an	8-bit text string, foreground	XDrawString
/disable or enable	access control	XSetAccessControl
XAddHost: add a host to the	access control list	XAddHost
add multiple hosts to the	access control list XAddHosts:	XAddHosts
/remove a host from the	access control list	XRemoveHost
/remove multiple hosts from the	access control list	XRemoveHosts
deny/ XEnableAccessControl: use	access control list to allow or	XEnableAccessControl
XDisableAccessControl: allow	access from any host	XDisableAccessControl
/obtain a list of hosts having	access to this display	XListHosts
XActivateScreenSaver:	activate screen blanking	XActivateScreenSaver
release the keyboard from an	active grab XUngrabKeyboard:	XUngrabKeyboard
release the pointer from an	active grab XUngrabPointer:	XUngrabPointer
/change the parameters of an	active pointer grab	XChangeActivePointerGrab
pixel value in an/ XAddPixel:	add a constant value to every	XAddPixel
list XAddHost:	add a host to the access control	XAddHost

XInsertModifiermapEntry:	add a new entry to an/	XInsertModifiermapEntry
XUnionRectWithRegion:	add a rectangle to a region	XUnionRectWithRegion
a/ XrmQPutStringResource:	add a resource specification to	XrmQPutStringResource
a resource/ XrmPutLineResource:	add a resource specification to	XrmPutLineResource
with/ XrmPutStringResource:	add a resource specification	XrmPutStringResource
save-set XAddToSaveSet:	add a window to the client's	XAddToSaveSet
control list XAddHosts:	add multiple hosts to the access	XAddHosts
the client's/ XChangeSaveSet:	add or remove a subwindow from	XChangeSaveSet
XrmUniqueQuark:	allocate a new quark	XrmUniqueQuark
from color/ XAllocNamedColor:	allocate a read-only colorcell	XAllocNamedColor
cell with closest/ XAllocColor:	allocate a read-only colormap	XAllocColor
XAllocClassHint:	allocate an XClassHint structure	XAllocClassHint
XAllocIconSize:	allocate an XIconSize structure	XAllocIconSize
XAllocSizeHints:	allocate an XSizeHints structure	XAllocSizeHints
XAllocStandardColormap:	allocate an XStandardColormap/	XAllocStandardColormap
XAllocWMHints:	allocate an XWMHints structure	XAllocWMHints
structure XCreateImage:	allocate memory for an XImage	XCreateImage
freed Xpermalloc:	allocate memory never to be	Xpermalloc
XAllocColorPlanes:	allocate read/write/	XAllocColorPlanes
colorcells XAllocColorCells:	allocate read/write (nonshared)	XAllocColorCells
XFree: free specified memory	allocated by an Xlib function	XFree
XFreeFontPath: free the memory	allocated by XGetFontPath	XFreeFontPath
XFreeFontNames: free the memory	allocated by XListFonts.	XFreeFontNames
XFreeFontInfo: free the memory	allocated by XListFontsWithInfo	XFreeFontInfo
XFreeExtensionList: free memory	allocated for a list of/	XFreeExtensionList
table. /free the memory	allocated for an association	XDestroyAssocTable
XDisableAccessControl:	allow access from any host	XDisableAccessControl
/use access control list to	allow or deny connection/	XEnableAccessControl
colormap; install default if not	already installed /uninstall a	XUninstallColormap
XLoadFont: load a font if not	already loaded; get font ID	XLoadFont
contents of one database into	another /merge the	XrmMergeDatabases
subtract one region from	another XSubtractRegion:	XSubtractRegion
system from one window to	another /change the coordinate	XTranslateCoordinates
/move the pointer to	another point on the screen	XWarpPointer
/insert a window between	another window and its parent	XReparentWindow
into a drawable with depth,	applying pixel values /drawable	XCopyPlane
/convert a keysym to the	appropriate keycode	XKeysymToKeycode
XFillArc: fill an	arc	XFillArc
XDrawArc: draw an	arc fitting inside a rectangle	XDrawArc
XSetArcMode: set the	arc mode in a graphics context	XSetArcMode
XDrawArcs: draw multiple	arcs	XDrawArcs
XFillArcs: fill multiple	arcs	XFillArcs
fill a rectangular	area XFillRectangle:	XFillRectangle
XClearArea: clear a rectangular	area in a window	XClearArea
XCopyArea: copy an	area of a drawable	XCopyArea
fill multiple rectangular	areas XFillRectangles:	XFillRectangles
database from command line	arguments /load a resource	XrmParseCommand
XA_WM_COMMAND atom (command line	arguments) XSetCommand: set the	XSetCommand
properties in the properties	array /rotate	XRotateWindowProperties
/obtain RGB values for an	array of colorcells	XQueryColors
/look up RGB values from	ASCII color name or translate/	XParseColor
/map a key event to	ASCII string, keysym, and/	XLookupString
XDefineCursor:	assign a cursor to a window	XDefineCursor
the window manager XStoreName:	assign a name to a window for	XStoreName
/deallocate storage	associated with a region	XDestroyRegion
/change a property	associated with a window	XChangeProperty
XDestroyImage: deallocate memory	associated with an image	XDestroyImage
/the GContext (resource ID)	associated with the specified/	XGContextFromGC

/report the display name (when	connection to a display fails)	XDisplayName
XNoOp: send a NoOp to exercise	connection with the server	XNoOp
value in an/ XAddPixel: add a	constant value to every pixel	XAddPixel
drawable into/ XGetImage: place	contents of a rectangle from	XGetImage
XrmMergeDatabases: merge the	contents of one database into/	XrmMergeDatabases
components of a given graphics	context XChangeGC: change the	XChangeGC
XCopyGC: copy a graphics	context	XCopyGC
context manager (not graphics	context) /get data from the	XFindContext
XFreeGC: free a graphics	context	XFreeGC
with the specified graphics	context /ID) associated	XGContextFromGC
and context type (not graphics	context) /to a window	XSaveContext
set the arc mode in a graphics	context XSetArcMode:	XSetArcMode
pixel value in a graphics	context /set the background	XSetBackground
clip_mask pixmap in a graphics	context XSetClipMask: set	XSetClipMask
the clip origin in a graphics	context XSetClipOrigin: set	XSetClipOrigin
of line dashes in a graphics	context /set a pattern	XSetDashes
set the fill rule in a graphics	context XSetFillRule:	XSetFillRule
set the fill style in a graphics	context XSetFillStyle:	XSetFillStyle
the current font in a graphics	context XSetFont: set	XSetFont
pixel value in a graphics	context /set the foreground	XSetForeground
logical operation in a graphics	context /set the bitwise	XSetFunction
component in a graphics	context /the graphics_exposures	XSetGraphicsExposures
drawing components in a graphics	context /set the line	XSetLineAttributes
set the plane mask in a graphics	context XSetPlaneMask:	XSetPlaneMask
and plane mask in a graphics	context /logical function,	XSetState
set the stipple in a graphics	context XSetStipple:	XSetStipple
the subwindow mode in a graphics	context XSetSubwindowMode: set	XSetSubwindowMode
set the fill tile in a graphics	context XSetTile:	XSetTile
origin in a graphics	context /set the tile/stipple	XSetTSOrigin
a new context ID (not graphics	context) XUniqueContext: create	XUniqueContext
and/ XDeleteContext: delete a	context entry for a given window	XDeleteContext
XCreateGC: create a new graphics	context for a given screen with/	XCreateGC
XUniqueContext: create a new	context ID (not graphics/	XUniqueContext
XFindContext: get data from the	context manager (not graphics/	XFindContext
/change clip_mask in a graphics	context to a list of rectangles	XSetClipRectangles
/set clip_mask of the graphics	context to the specified region	XSetRegion
/corresponding to a window and	context type (not graphics/	XSaveContext
disable or enable access	control XSetAccessControl:	XSetAccessControl
mapping of modifier keys (Shift,	Control, etc.) /obtain a	XGetModifierMapping
to be used as modifiers (Shift,	Control, etc.) /set keycodes	XSetModifierMapping
XBell: ring the bell	(Control G)	XBell
add a host to the access	control list XAddHost:	XAddHost
add multiple hosts to the access	control list XAddHosts:	XAddHosts
remove a host from the access	control list XRemoveHost:	XRemoveHost
multiple hosts from the access	control list /remove	XRemoveHosts
XEnableAccessControl: use access	control list to allow or deny/	XEnableAccessControl
and pointer/ XAllowEvents:	control the behavior of keyboard	XAllowEvents
XrmStringToBindingQuarkList:	convert a key string to a/	XrmStringToBindingQuarkList
list XrmStringToQuarkList:	convert a key string to a quark	XrmStringToQuarkList
XKeycodeToKeysym:	convert a keycode to a keysym	XKeycodeToKeysym
a keysym XStringToKeysym:	convert a keysym name string to	XStringToKeysym
string XKeysymToString:	convert a keysym symbol to a	XKeysymToString
appropriate/ XKeysymToKeycode:	convert a keysym to the	XKeysymToKeycode
XrmQuarkToString:	convert a quark to a string	XrmQuarkToString
XrmStringToQuark:	convert a string to a quark	XrmStringToQuark
window to another /change the	coordinate system from one	XTranslateCoordinates
colormap/ XCopyColormapAndFree:	copy a colormap and return a new	XCopyColormapAndFree
XCopyGC:	copy a graphics context	XCopyGC

a location within/ XGetSubImage:	copy a rectangle in drawable to	XGetSubImage
drawable into a/ XCopyPlane:	copy a single plane of a	XCopyPlane
XCopyArea:	copy an area of a drawable	XCopyArea
XLookupKeysym: get the keysym	corresponding to a keycode in/	XLookupKeysym
XSaveContext: save a data value	corresponding to a window and/	XSaveContext
format/ XCreateBitmapFromData:	create a bitmap from X11 bitmap	XCreateBitmapFromData
XCreateColormap:	create a colormap.	XCreateColormap
XCreateGlyphCursor:	create a cursor from font glyphs	XCreateGlyphCursor
standard/ XCreateFontCursor:	create a cursor from the	XCreateFontCursor
XCreatePixmapCursor:	create a cursor from two bitmaps	XCreatePixmapCursor
XrmGetStringDatabase:	create a database from a string	XrmGetStringDatabase
mapping/ XNewModifiermap:	create a keyboard modifier	XNewModifiermap
(X10) XCreateAssocTable:	create a new association table	XCreateAssocTable
graphics/ XUniqueContext:	create a new context ID (not	XUniqueContext
XCreateRegion:	create a new empty region	XCreateRegion
for a given screen/ XCreateGC:	create a new graphics context	XCreateGC
XCreatePixmap:	create a pixmap	XCreatePixmap
XCreatePixmapFromBitmapData:	create a pixmap with depth from/	XCreatePixmapFromBitmapDat
an image XSubImage:	create a subimage from part of	XSubImage
attributes XCreateWindow:	create a window and set	XCreateWindow
association table XMakeAssoc:	create an entry in an	XMakeAssoc
window XCreateSimpleWindow:	create an unmapped InputOutput	XCreateSimpleWindow
XGetWindowAttributes: obtain the	current attributes of window	XGetWindowAttributes
context XSetFont: set the	current font in a graphics	XSetFont
XGetFontPath: get the	current font search path	XGetFontPath
XGetGeometry: obtain the	current geometry of drawable	XGetGeometry
XGetInputFocus: return the	current keyboard focus window	XGetInputFocus
/obtain a list of the	current keyboard preferences	XGetKeyboardControl
XQueryPointer: get the	current pointer location	XQueryPointer
XGetPointerControl: get the	current pointer preferences	XGetPointerControl
XGetScreenSaver: get the	current screen saver parameters	XGetScreenSaver
/obtain a bit vector for the	current state of the keyboard	XQueryKeymap
XFreeCursor: release a	cursor	XFreeCursor
change the color of a	cursor XRecolorCursor:	XRecolorCursor
a cursor from the standard	cursor font /create	XCreateFontCursor
XUndefineCursor: disassociate a	cursor from a window	XUndefineCursor
XCreateGlyphCursor: create a	cursor from font glyphs	XCreateGlyphCursor
XCreateFontCursor: create a	cursor from the standard cursor/	XCreateFontCursor
XCreatePixmapCursor: create a	cursor from two bitmaps	XCreatePixmapCursor
get the closest supported	cursor sizes XQueryBestCursor:	XQueryBestCursor
/obtain the "best" supported	cursor, tile, or stipple size	XQueryBestSize
XDefineCursor: assign a	cursor to a window	XDefineCursor
X10) XDraw: draw a polyline or	curve between vertex list (from	XDraw
X10) /draw a filled polygon or	curve from vertex list (from	XDrawFilled
XFetchBuffer: return data from a	cut buffer	XFetchBuffer
XStoreBuffer: store data in a	cut buffer	XStoreBuffer
XFetchBytes: return data from	cut buffer 0	XFetchBytes
XStoreBytes: store data in	cut buffer 0	XStoreBytes
XRotateBuffers: rotate the	cut buffers	XRotateBuffers
/set a pattern of line	dashes in a graphics context	XSetDashes
a bitmap from X11 bitmap format	data /create	XCreateBitmapFromData
a pixmap with depth from bitmap	data. /create	XCreatePixmapFromBitmapData
specified/ /free the in-memory	data associated with the	XFreeStringList
XFetchBuffer: return	data from a cut buffer	XFetchBuffer
XLookUpAssoc: obtain	data from an association table	XLookUpAssoc
XFetchBytes: return	data from cut buffer 0	XFetchBytes
(not graphics/ XFindContext: get	data from the context manager	XFindContext
XStoreBuffer: store	data in a cut buffer	XStoreBuffer

XStoreBytes: store	data in cut buffer 0	XStoreBytes
window and/ XSaveContext: save a	data value corresponding to a	XSaveContext
option value from the resource	database /extract an	XGetDefault
error messages from the error	database /obtain	XGetErrorDatabaseText
destroy a resource	database. XrmDestroyDatabase:	XrmDestroyDatabase
specification to a resource	database /add a resource	XrmPutLineResource
specification into a resource	database /store a resource	XrmPutResource
XrmGetFileDatabase: retrieve a	database from a file	XrmGetFileDatabase
XrmGetStringDatabase: create a	database from a string	XrmGetStringDatabase
XrmParseCommand: load a resource	database from command line/	XrmParseCommand
/store a resource	database in a file	XrmPutFileDatabase
/merge the contents of one	database into another	XrmMergeDatabases
/return a list of	database levels	XrmQGetSearchList
XLookupColor: get	database RGB values and closest/	XLookupColor
/a resource specification to a	database using a quark resource/	XrmQPutStringResource
a resource specification into a	database using quarks /store	XrmQPutResource
with an image XDestroyImage:	deallocate memory associated	XDestroyImage
with a region XDestroyRegion:	deallocate storage associated	XDestroyRegion
or disable synchronization for	debugging XSynchronize: enable	XSynchronize
a colormap and install the	default colormap /delete	XFreeColormap
given user geometry string and	default geometry /geometry	XGeometry
/uninstall a colormap; install	default if not already installed	XUninstallColormap
the default/ XFreeColormap:	delete a colormap and install	XFreeColormap
given window/ XDeleteContext:	delete a context entry for a	XDeleteContext
XDeleteProperty:	delete a window property	XDeleteProperty
association/ XDeleteAssoc:	delete an entry from an	XDeleteAssoc
XDeleteModifiermapEntry:	delete an entry from an/	XDeleteModifiermapEntry
access control list to allow or	deny connection requests /use	XEnableAccessControl
that matches the desired	depth and class /information	XMatchVisualInfo
a drawable into a drawable with	depth, applying pixel values /of	XCopyPlane
/create a pixmap with	depth from bitmap data.	XCreatePixmapFromBitmapData
/for a given screen with the	depth of the specified drawable	XCreateGC
XListDepths: determine the	depths available on a given/	XListDepths
XGetErrorText: obtain a	description of error code	XGetErrorText
information that matches the	desired depth and class /visual	XMatchVisualInfo
remaining/ XKillClient:	destroy a client or its	XKillClient
XrmDestroyDatabase:	destroy a resource database.	XrmDestroyDatabase
XDestroyWindow: unmap and	destroy a window and all/	XDestroyWindow
window XDestroySubwindows:	destroy all subwindows of a	XDestroySubwindows
modifier/ XFreeModifiermap:	destroy and free a keyboard	XFreeModifiermap
region XPointInRegion:	determine if a point is inside a	XPointInRegion
in a region XRectInRegion:	determine if a rectangle resides	XRectInRegion
XEmptyRegion:	determine if a region is empty	XEmptyRegion
the same size,/ XEqualRegion:	determine if two regions have	XEqualRegion
on a given screen XListDepths:	determine the depths available	XListDepths
XXorRegion: calculate the	difference between the union and/	XXorRegion
XSetAccessControl:	disable or enable access control	XSetAccessControl
XSynchronize: enable or	disable synchronization for/	XSynchronize
window XUndefineCursor:	disassociate a cursor from a	XUndefineCursor
an X server and/ XCloseDisplay:	disconnect a client program from	XCloseDisplay
XDrawSegments: draw multiple	disjoint lines	XDrawSegments
read a bitmap from	disk XReadBitmapFile:	XReadBitmapFile
program from an X server and	display /disconnect a client	XCloseDisplay
of hosts having access to this	display /obtain a list	XListHosts
XFlush: flush the request buffer	(display all queued requests)	XFlush
name (when connection to a	display fails) /the display	XDisplayName
a/ XDisplayName: report the	display name (when connection to	XDisplayName
XSetIconName: set the name to be	displayed in a window's icon	XSetIconName

procedure XIfEvent: wait for	event matched in predicate	XIfEvent
window and/ /remove the next	event matching both passed	XCheckWindowEvent
XNextEvent: get the next	event of any type or window	XNextEvent
the number of events in the	event queue /check	XEventsQueued
XCheckIfEvent: check the	event queue for a matching event	XCheckIfEvent
XMaskEvent: remove the next	event that matches mask	XMaskEvent
XCheckMaskEvent: remove the next	event that matches mask; don't/	XCheckMaskEvent
XWindowEvent: remove the next	event that matches the specified/	XWindowEvent
and/ XLookupString: map a key	event to ASCII string, keysym,	XLookupString
next event in queue that matches	event type; don't wait /the	XCheckTypedEvent
window XSelectInput: select the	event types to be sent to a	XSelectInput
the queue XPeekEvent: get an	event without removing it from	XPeekEvent
the number of pending input	events /buffer and return	XPending
request buffer and wait for all	events and errors to be/ /the	XSync
history/ XGetMotionEvents: get	events from pointer motion	XGetMotionEvents
/check the number of	events in the event queue	XEventsQueued
/behavior of keyboard and pointer	events when these resources are/	XAllowEvents
server XNoOp: send a NoOp to	exercise connection with the	XNoOp
XShrinkRegion: reduce or	expand the size of a region	XShrinkRegion
XQueryExtension: get	extension information	XQueryExtension
for a list of installed	extensions /memory allocated	XFreeExtensionList
Xlib and/ /return a list of all	extensions to X supported by	XListExtensions
resource database XGetDefault:	extract an option value from the	XGetDefault
(when connection to a display	fails) /report the display name	XDisplayName
XQueryBestTile: obtain the	fastest supported fill tile/	XQueryBestTile
XQueryBestStipple: obtain the	fastest supported stipple shape	XQueryBestStipple
retrieve a database from a	file XrmGetFileDatabase:	XrmGetFileDatabase
store a resource database in a	file XrmPutFileDatabase:	XrmPutFileDatabase
write a bitmap to a	file XWriteBitmapFile:	XWriteBitmapFile
XFillPolygon:	fill a polygon	XFillPolygon
XFillRectangle:	fill a rectangular area	XFillRectangle
XFillArc:	fill an arc	XFillArc
XLoadQueryFont: load a font and	fill information structure	XLoadQueryFont
XFillArcs:	fill multiple arcs	XFillArcs
XFillRectangles:	fill multiple rectangular areas	XFillRectangles
XSetFillRule: set the	fill rule in a graphics context	XSetFillRule
XSetFillStyle: set the	fill style in a graphics context	XSetFillStyle
XSetTile: set the	fill tile in a graphics context	XSetTile
obtain the fastest supported	fill tile shape XQueryBestTile:	XQueryBestTile
vertex list/ XDrawFilled: draw a	filled polygon or curve from	XDrawFilled
structures that/ XGetVisualInfo:	find the visual information	XGetVisualInfo
XDrawArc: draw an arc	fitting inside a rectangle	XDrawArc
/obtain the RGB values and	flags for a specified colorcell	XQueryColor
return the number of/ XPending:	flush the request buffer and	XPending
wait for all events and/ XSync:	flush the request buffer and	XSync
(display all queued/ XFlush:	flush the request buffer	XFlush
return the current keyboard	focus window XGetInputFocus:	XGetInputFocus
XSetInputFocus: set the keyboard	focus window	XSetInputFocus
cursor from the standard cursor	font /create a	XCreateFontCursor
information about a loaded	font XQueryFont: return	XQueryFont
XUnloadFont: unload a	font.	XUnloadFont
XLoadQueryFont: load a	font and fill information/	XLoadQueryFont
font/ XFreeFont: unload a	font and free storage for the	XFreeFont
create a cursor from	font glyphs XCreateGlyphCursor:	XCreateGlyphCursor
font if not already loaded; get	font ID XLoadFont: load a	XLoadFont
font ID XLoadFont: load a	font if not already loaded; get	XLoadFont
XSetFont: set the current	font in a graphics context	XSetFont
query the server for string and	font metrics XQueryTextExtents:	XQueryTextExtents

XTextExtents: get string and	font metrics locally	XTextExtents
/query the server for string and	font metrics of a 16-bit/	XQueryTextExtents16
XTextExtents16: get string and	font metrics of a 16-bit/	XTextExtents16
return a list of the available	font names XListFonts:	XListFonts
XGetFontProperty: get a	font property given its atom	XGetFontProperty
XGetFontPath: get the current	font search path	XGetFontPath
XSetFontPath: set the	font search path	XSetFontPath
a font and free storage for the	font structure /unload	XFreeFont
and information about loaded	fonts /obtain the names	XListFontsWithInfo
function,/ XSetState: set the	foreground, background, logical	XSetState
draw an 8-bit text string,	foreground only XDrawString:	XDrawString
XSetForeground: set the	foreground pixel value in a/	XSetForeground
/create a bitmap from X11 bitmap	format data	XCreateBitmapFromData
the atom type and property	format for a window /obtain	XGetWindowProperty
/obtain the supported pixmap	formats for a given server	XListPixmapFormats
XFreeGC:	free a graphics context	XFreeGC
XFreeModifiermap: destroy and	free a keyboard modifier mapping/	XFreeModifiermap
XFreePixmap:	free a pixmap ID	XFreePixmap
XFreeColors:	free colormap cells or planes	XFreeColors
of/ XFreeExtensionList:	free memory allocated for a list	XFreeExtensionList
by an Xlib function XFree:	free specified memory allocated	XFree
XFreeFont: unload a font and	free storage for the font/	XFreeFont
associated/ XFreeStringList:	free the in-memory data	XFreeStringList
XGetFontPath XFreeFontPath:	free the memory allocated by	XFreeFontPath
XListFonts. XFreeFontNames:	free the memory allocated by	XFreeFontNames
XFreeFontInfo:	free the memory allocated by/	XFreeFontInfo
association/ XDestroyAssocTable:	free the memory allocated for an	XDestroyAssocTable
allocate memory never to be	freed Xpermalloc:	Xpermalloc
memory allocated by an Xlib	function XFree: free specified	XFree
/foreground, background, logical	function, and plane mask in a/	XSetState
XSetAfterFunction: set a	function called after all Xlib/	XSetAfterFunction
a function called after all Xlib	functions /set	XSetAfterFunction
XBell: ring the bell (Control	G)	XBell
of a given GC from Xlib's	GC cache /obtain components	XGetGCValues
/obtain components of a given	GC from Xlib's GC cache	XGetGCValues
XGContextFromGC: obtain the	GContext (resource ID)/	XGContextFromGC
XPolygonRegion:	generate a region from points	XPolygonRegion
standard window/ XParseGeometry:	generate position and size from	XParseGeometry
enclosing a region XClipBox:	generate the smallest rectangle	XClipBox
user geometry string and default	geometry /window geometry given	XGeometry
XGeometry: calculate window	geometry given user geometry/	XGeometry
XWMGeometry: obtain a window's	geometry information	XWMGeometry
XGetGeometry: obtain the current	geometry of drawable	XGetGeometry
and size from standard window	geometry string /position	XParseGeometry
/window geometry given user	geometry string and default/	XGeometry
atom XGetFontProperty:	get a font property given its	XGetFontProperty
XListInstalledColormaps:	get a list of installed/	XListInstalledColormaps
class as/ XrmGetResource:	get a resource from name and	XrmGetResource
and class as/ XrmQGetResource:	get a resource value using name	XrmQGetResource
given its atom XGetAtomName:	get a string name for a property	XGetAtomName
property) XFetchName:	get a window's name (XA_WM_NAME	XFetchName
predicate/ XPeekIfEvent:	get an event matched by	XPeekIfEvent
from the queue XPeekEvent:	get an event without removing it	XPeekEvent
manager (not/ XFindContext:	get data from the context	XFindContext
closest/ XLookupColor:	get database RGB values and	XLookupColor
history/ XGetMotionEvents:	get events from pointer motion	XGetMotionEvents
XQueryExtension:	get extension information	XQueryExtension
a font if not already loaded;	get font ID XLoadFont: load	XLoadFont

XGetIconSizes:	get preferred icon sizes	XGetIconSizes
locally XTextExtents:	get string and font metrics	XTextExtents
16-bit/ XTextExtents16:	get string and font metrics of a	XTextExtents16
sizes XQueryBestCursor:	get the closest supported cursor	XQueryBestCursor
XGetFontPath:	get the current font search path	XGetFontPath
XQueryPointer:	get the current pointer location	XQueryPointer
preferences XGetPointerControl:	get the current pointer	XGetPointerControl
parameters XGetScreenSaver:	get the current screen saver	XGetScreenSaver
a keycode in/ XLookupKeysym:	get the keysym corresponding to	XLookupKeysym
an icon XGetIconName:	get the name to be displayed in	XGetIconName
or window XNextEvent:	get the next event of any type	XNextEvent
XGetPointerMapping:	get the pointer button mapping	XGetPointerMapping
window XListProperties:	get the property list for a	XListProperties
window in/ XGetNormalHints:	get the size hints property of a	XGetNormalHints
property XGetStandardColormap:	get the standard colormap	XGetStandardColormap
16-bit character/ XTextWidth16:	get the width in pixels of a	XTextWidth16
8-bit character/ XTextWidth:	get the width in pixels of an	XTextWidth
a window XGetClassHint:	get the XA_WM_CLASS property of	XGetClassHint
property/ XGetTransientForHint:	get the XA_WM_TRANSIENT_FOR	XGetTransientForHint
create a cursor from font	glyphs XCreateGlyphCursor:	XCreateGlyphCursor
parameters of an active pointer	grab /change the	XChangeActivePointerGrab
release a button from a passive	grab XUngrabButton:	XUngrabButton
release a key from a passive	grab XUngrabKey:	XUngrabKey
the keyboard from an active	grab XUngrabKeyboard: release	XUngrabKeyboard
the pointer from an active	grab XUngrabPointer: release	XUngrabPointer
release the server from	grab XUngrabServer:	XUngrabServer
XGrabKey:	grab a key	XGrabKey
XGrabButton:	grab a pointer button	XGrabButton
XGrabKeyboard:	grab the keyboard	XGrabKeyboard
XGrabPointer:	grab the pointer	XGrabPointer
XGrabServer:	grab the server	XGrabServer
events when these resources are	grabbed /of keyboard and pointer	XAllowEvents
change the components of a given	graphics context XChangeGC:	XChangeGC
XCopyGC: copy a	graphics context	XCopyGC
from the context manager (not	graphics context) /get data	XFindContext
XFreeGC: free a	graphics context	XFreeGC
associated with the specified	graphics context /(resource ID)	XGContextFromGC
a window and context type (not	graphics context) /to	XSaveContext
set the arc mode in a	graphics context XSetArcMode:	XSetArcMode
the background pixel value in a	graphics context /set	XSetBackground
set clip_mask pixmap in a	graphics context XSetClipMask:	XSetClipMask
/set the clip origin in a	graphics context	XSetClipOrigin
a pattern of line dashes in a	graphics context /set	XSetDashes
set the fill rule in a	graphics context XSetFillRule:	XSetFillRule
set the fill style in a	graphics context XSetFillStyle:	XSetFillStyle
set the current font in a	graphics context XSetFont:	XSetFont
the foreground pixel value in a	graphics context /set	XSetForeground
bitwise logical operation in a	graphics context /set the	XSetFunction
/component in a	graphics context	XSetGraphicsExposures
the line drawing components in a	graphics context /set	XSetLineAttributes
set the plane mask in a	graphics context XSetPlaneMask:	XSetPlaneMask
function, and plane mask in a	graphics context /logical	XSetState
set the stipple in a	graphics context XSetStipple:	XSetStipple
/set the subwindow mode in a	graphics context	XSetSubwindowMode
XSetTile: set the fill tile in a	graphics context	XSetTile
set the tile/stipple origin in a	graphics context XSetTSOrigin:	XSetTSOrigin
/create a new context ID (not	graphics context)	XUniqueContext
screen/ XCreateGC: create a new	graphics context for a given	XCreateGC

XDrawArc: draw an arc fitting	inside a rectangle	XDrawArc
determine if a point is	inside a region XPointInRegion:	XPointInRegion
XInstallColormap:	install a colormap	XInstallColormap
installed /uninstall a colormap;	install default if not already	XUninstallColormap
/delete a colormap and	install the default colormap	XFreeColormap
install default if not already	installed /uninstall a colormap;	XUninstallColormap
/get a list of	installed colormaps	XListInstalledColormaps
memory allocated for a list of	installed extensions /free	XFreeExtensionList
XIntersectRegion: compute the	intersection of two regions	XIntersectRegion
difference between the union and	intersection of two regions /the	XXorRegion
XGrabKey: grab a	key	XGrabKey
the keyboard preferences such as	key click /change	XChangeKeyboardControl
keysym,/ XLookupString: map a	key event to ASCII string,	XLookupString
XUngrabKey: release a	key from a passive grab	XUngrabKey
a quark list /convert a	key string to a binding list and	XrmStringToBindingQuarkList
XrmStringToQuarkList: convert a	key string to a quark list	XrmStringToQuarkList
XGrabKeyboard: grab the	keyboard	XGrabKeyboard
for the current state of the	keyboard /obtain a bit vector	XQueryKeymap
these/ /control the behavior of	keyboard and pointer events when	XAllowEvents
XAutoRepeatOff: turn off the	keyboard auto-repeat keys	XAutoRepeatOff
XAutoRepeatOn: turn on the	keyboard auto-repeat keys	XAutoRepeatOn
/return the current	keyboard focus window	XGetInputFocus
XSetInputFocus: set the	keyboard focus window	XSetInputFocus
XUngrabKeyboard: release the	keyboard from an active grab	XUngrabKeyboard
/change the	keyboard mapping	XChangeKeyboardMapping
structure /destroy and free a	keyboard modifier mapping	XFreeModifiermap
XNewModifiermap: create a	keyboard modifier mapping/	XNewModifiermap
/obtain a list of the current	keyboard preferences	XGetKeyboardControl
click /change the	keyboard preferences such as key	XChangeKeyboardControl
a keysym to the appropriate	keycode /convert	XKeysymToKeycode
the keysym corresponding to a	keycode in structure /get	XLookupKeysym
XKeycodeToKeysym: convert a	keycode to a keysym	XKeycodeToKeysym
XRefreshKeyboardMapping: read	keycode-keysym mapping from/	XRefreshKeyboardMapping
return symbols for	keycodes XGetKeyboardMapping:	XGetKeyboardMapping
/obtain the range of legal	keycodes for a server	XDisplayKeycodes
XSetModifierMapping: set	keycodes to be used as modifiers/	XSetModifierMapping
off the keyboard auto-repeat	keys XAutoRepeatOff: turn	XAutoRepeatOff
turn on the keyboard auto-repeat	keys XAutoRepeatOn:	XAutoRepeatOn
/obtain a mapping of modifier	keys (Shift, Control, etc.)	XGetModifierMapping
convert a keycode to a	keysym XKeycodeToKeysym:	XKeycodeToKeysym
a keysym name string to a	keysym XStringToKeysym: convert	XStringToKeysym
/map a key event to ASCII string,	keysym, and ComposeStatus	XLookupString
keycode/ XLookupKeysym: get the	keysym corresponding to a	XLookupKeysym
XStringToKeysym: convert a	keysym name string to a keysym	XStringToKeysym
XKeysymToString: convert a	keysym symbol to a string	XKeysymToString
XRebindKeysym: rebind a	keysym to a string for client	XRebindKeysym
XKeysymToKeycode: convert a	keysym to the appropriate/	XKeysymToKeycode
/obtain the range of	legal keycodes for a server	XDisplayKeycodes
return a list of database	levels XrmQGetSearchList:	XrmQGetSearchList
a resource database from command	line arguments /load	XrmParseCommand
the XA_WM_COMMAND atom (command	line arguments) /set	XSetCommand
XDrawLine: draw a	line between two points	XDrawLine
XSetDashes: set a pattern of	line dashes in a graphics/	XSetDashes
XSetLineAttributes: set the	line drawing components in a/	XSetLineAttributes
draw multiple connected	lines. XDrawLines:	XDrawLines
draw multiple disjoint	lines XDrawSegments:	XDrawSegments
add a host to the access control	list XAddHost:	XAddHost
hosts to the access control	list XAddHosts: add multiple	XAddHosts

the next event that matches	mask; don't wait /remove	XCheckMaskEvent
both passed window and passed	mask; don't wait /event matching	XCheckWindowEvent
XSetPlaneMask: set the plane	mask in a graphics context	XSetPlaneMask
/logical function, and plane	mask in a graphics context	XSetState
/information structures that	match the specified template	XGetVisualInfo
XPeekIfEvent: get an event	matched by predicate procedure/	XPeekIfEvent
XIfEvent: wait for event	matched in predicate procedure	XIfEvent
/the next event in queue that	matches event type; don't wait	XCheckTypedEvent
remove the next event that	matches mask XMaskEvent:	XMaskEvent
/remove the next event that	matches mask; don't wait	XCheckMaskEvent
/the visual information that	matches the desired depth and/	XMatchVisualInfo
/remove the next event that	matches the specified mask and/	XWindowEvent
passed/ /remove the next event	matching both passed window and	XCheckWindowEvent
check the event queue for a	matching event XCheckIfEvent:	XCheckIfEvent
/return the next event in queue	matching type and window	XCheckTypedWindowEvent
function XFree: free specified	memory allocated by an Xlib	XFree
XFreeFontPath: free the	memory allocated by XGetFontPath	XFreeFontPath
XFreeFontNames: free the	memory allocated by XListFonts.	XFreeFontNames
XFreeFontInfo: free the	memory allocated by/	XFreeFontInfo
XFreeExtensionList: free	memory allocated for a list of/	XFreeExtensionList
XDestroyAssocTable: free the	memory allocated for an/	XDestroyAssocTable
XDestroyImage: deallocate	memory associated with an image	XDestroyImage
XCreateImage: allocate	memory for an XImage structure	XCreateImage
Xpermalloc: allocate	memory never to be freed	Xpermalloc
database/ XrmMergeDatabases:	merge the contents of one	XrmMergeDatabases
/obtain error	messages from the error database	XGetErrorDatabaseText
the server for string and font	metrics /query	XQueryTextExtents
get string and font	metrics locally XTextExtents:	XTextExtents
/the server for string and font	metrics of a 16-bit character/	XQueryTextExtents16
string,/ /get string and font	metrics of a 16-bit character	XTextExtents16
XSetStandardProperties: set the	minimum set of properties for/	XSetStandardProperties
XSetArcMode: set the arc	mode in a graphics context	XSetArcMode
/set the subwindow	mode in a graphics context	XSetSubwindowMode
/change the close down	mode of a client	XSetCloseDownMode
etc.) /obtain a mapping of	modifier keys (Shift, Control,	XGetModifierMapping
/destroy and free a keyboard	modifier mapping structure	XFreeModifiermap
/create a keyboard	modifier mapping structure	XNewModifiermap
/set keycodes to be used as	modifiers (Shift, Control, etc.)	XSetModifierMapping
/get events from pointer	motion history buffer	XGetMotionEvents
XMoveWindow:	move a window	XMoveWindow
point on the/ XWarpPointer:	move the pointer to another	XWarpPointer
XDrawArcs: draw	multiple arcs	XDrawArcs
XFillArcs: fill	multiple arcs	XFillArcs
XDrawLines: draw	multiple connected lines.	XDrawLines
XDrawSegments: draw	multiple disjoint lines	XDrawSegments
control/ XRemoveHosts: remove	multiple hosts from the access	XRemoveHosts
control list XAddHosts: add	multiple hosts to the access	XAddHosts
XDrawPoints: draw	multiple points.	XDrawPoints
/draw the outlines of	multiple rectangles	XDrawRectangles
XFillRectangles: fill	multiple rectangular areas	XFillRectangles
a read-only colorcell from color	name XAllocNamedColor: allocate	XAllocNamedColor
RGB values from color	name /closest hardware-supported	XLookupColor
a read/write colorcell by color	name /set RGB values of	XStoreNamedColor
/get a resource value using	name and class as quarks	XrmQGetResource
/get a resource from	name and class as strings	XrmGetResource
database using a quark resource	name and string value /to a	XrmQPutStringResource
with separate resource	name and value /specification	XrmPutStringResource
atom XGetAtomName: get a string	name for a property given its	XGetAtomName

ID) associated/ XGContextFromGC:	obtain the GContext (resource	XGContextFromGC
about/ XListFontsWithInfo:	obtain the names and information	XListFontsWithInfo
keycodes for/ XDisplayKeycodes:	obtain the range of legal	XDisplayKeycodes
for a specified/ XQueryColor:	obtain the RGB values and flags	XQueryColor
formats for/ XListPixmapFormats:	obtain the supported pixmap	XListPixmapFormats
Visual XVisualIDFromVisual:	obtain the visual ID from a	XVisualIDFromVisual
that matches/ XMatchVisualInfo:	obtain the visual information	XMatchVisualInfo
structure/ XGetRGBColormaps:	obtain the XStandardColormap	XGetRGBColormaps
turn the screen saver on or	off XForceScreenSaver:	XForceScreenSaver
keys XAutoRepeatOff: turn	off the keyboard auto-repeat	XAutoRepeatOff
two regions have the same size,	offset, and shape /determine if	XEqualRegion
XOffsetRegion: change	offset of a region	XOffsetRegion
an 8-bit text string, foreground	only XDrawString: draw	XDrawString
/set the bitwise logical	operation in a graphics context	XSetFunction
XGetDefault: extract an	option value from the resource/	XGetDefault
child to the top of the stacking	order /circulate the bottom	XCirculateSubwindowsDown
to the bottom of the stacking	order /circulate the top child	XCirculateSubwindowsUp
size, border width, or stacking	order /the window position,	XConfigureWindow
lower a window in the stacking	order XLowerWindow:	XLowerWindow
to the top of the stacking	order /raise a window	XRaiseWindow
/circulate the stacking	order of children up or down	XCirculateSubwindows
/change the stacking	order of siblings	XRestackWindows
XSetClipOrigin: set the clip	origin in a graphics context	XSetClipOrigin
/set the tile/stipple	origin in a graphics context	XSetTSOrigin
XDrawRectangle: draw an	outline of a rectangle	XDrawRectangle
XDrawRectangles: draw the	outlines of multiple rectangles	XDrawRectangles
XGetSelectionOwner: return the	owner of a selection	XGetSelectionOwner
XSetSelectionOwner: set the	owner of a selection	XSetSelectionOwner
get the current screen saver	parameters XGetScreenSaver:	XGetScreenSaver
grab /change the	parameters of an active pointer	XChangeActivePointerGrab
XSetScreenSaver: set the	parameters of the screen saver	XSetScreenSaver
between another window and its	parent /insert a window	XReparentWindow
return a list of children,	parent, and root XQueryTree:	XQueryTree
create a subimage from	part of an image XSubImage:	XSubImage
matching both passed window and	passed mask; don't wait /event	XCheckWindowEvent
/the next event matching both	passed window and passed mask;/	XCheckWindowEvent
release a button from a	passive grab XUngrabButton:	XUngrabButton
XUngrabKey: release a key from a	passive grab	XUngrabKey
get the current font search	path XGetFontPath:	XGetFontPath
set the font search	path XSetFontPath:	XSetFontPath
graphics/ XSetDashes: set a	pattern of line dashes in a	XSetDashes
buffer and return the number of	pending input events /request	XPending
repaint/ /change a window border	pixel value attribute and	XSetWindowBorder
window /set the background	pixel value attribute of a	XSetWindowBackground
XGetPixel: obtain a single	pixel value from an image	XGetPixel
context /set the background	pixel value in a graphics	XSetBackground
context /set the foreground	pixel value in a graphics	XSetForeground
/add a constant value to every	pixel value in an image	XAddPixel
XPutPixel: set a	pixel value in an image	XPutPixel
a drawable with depth, applying	pixel values /of a drawable into	XCopyPlane
XTextWidth16: get the width in	pixels of a 16-bit character/	XTextWidth16
XTextWidth: get the width in	pixels of an 8-bit character/	XTextWidth
XCreatePixmap: create a	pixmap	XCreatePixmap
draw an image on a window or	pixmap XPutImage:	XPutImage
server /obtain the supported	pixmap formats for a given	XListPixmapFormats
XFreePixmap: free a	pixmap ID	XFreePixmap
XSetClipMask: set clip_mask	pixmap in a graphics context	XSetClipMask
data. /create a	pixmap with depth from bitmap	XCreatePixmapFromBitmapData

read the window manager hints	property XGetWMHints:	XGetWMHints
read a window's XA_WM_ICON_NAME	property XGetWMIconName:	XGetWMIconName
read a window's XA_WM_NAME	property XGetWMName:	XGetWMName
a window's XA_WM_NORMAL_HINTS	property /read	XGetWMNormalHints
read a window's XA_WM_SIZE_HINTS	property XGetWMSizeHints:	XGetWMSizeHints
the value of the XA_WM_ICON_SIZE	property XSetIconSizes: set	XSetIconSizes
change the standard colormap	property XSetStandardColormap:	XSetStandardColormap
set a window's WM_CLIENT_MACHINE	property XSetWMClientMachine:	XSetWMClientMachine
a window's WM_COLORMAP_WINDOWS	property /set	XSetWMColormapWindows
set a window manager hints	property XSetWMHints:	XSetWMHints
set a window's XA_WM_ICON_NAME	property XSetWMIconName:	XSetWMIconName
set a window's XA_WM_NAME	property. XSetWMName:	XSetWMName
a window's XA_WM_NORMAL_HINTS	property XSetWMNormalHints: set	XSetWMNormalHints
set a window's WM_PROTOCOLS	property XSetWMProtocols:	XSetWMProtocols
set a window's WM_SIZE_HINTS	property XSetWMSizeHints:	XSetWMSizeHints
XChangeProperty: change a	property associated with a/	XChangeProperty
/set the XA_WM_TRANSIENT_FOR	property for a window	XSetTransientForHint
/obtain the atom type and	property format for a window	XGetWindowProperty
/get a string name for a	property given its atom	XGetAtomName
XGetFontProperty: get a font	property given its atom	XGetFontProperty
XListProperties: get the	property list for a window	XListProperties
/return an atom for a given	property name string	XInternAtom
/get the XA_WM_CLASS	property of a window	XGetClassHint
/get the XA_WM_TRANSIENT_FOR	property of a window	XGetTransientForHint
/set the XA_WM_CLASS	property of a window	XSetClassHint
state (not/ /get the size hints	property of a window in normal	XGetNormalHints
state (not/ /set the size hints	property of a window in normal	XSetNormalHints
/read the size hints	property of a zoomed window	XGetZoomHints
/set the size hints	property of a zoomed window	XSetZoomHints
XGetSizeHints: read any	property of type XA_SIZE_HINTS	XGetSizeHints
/set the value of any	property of type XA_SIZE_HINTS	XSetSizeHints
queue XPutBackEvent:	push an event back on the input	XPutBackEvent
convert a string to a	quark XrmStringToQuark:	XrmStringToQuark
XrmUniqueQuark: allocate a new	quark	XrmUniqueQuark
string to a binding list and a	quark list /convert a key	XrmStringToBindingQuarkList
/convert a key string to a	quark list	XrmStringToQuarkList
value /to a database using a	quark resource name and string	XrmQPutStringResource
XrmQuarkToString: convert a	quark to a string	XrmQuarkToString
value using name and class as	quarks /get a resource	XrmQGetResource
into a database using	quarks /a resource specification	XrmQPutResource
font metrics XQueryTextExtents:	query the server for string and	XQueryTextExtents
font/ XQueryTextExtents16:	query the server for string and	XQueryTextExtents16
number of events in the event	queue XEventsQueued: check the	XEventsQueued
without removing it from the	queue XPeekEvent: get an event	XPeekEvent
without removing it from the	queue /by predicate procedure	XPeekIfEvent
push an event back on the input	queue XPutBackEvent:	XPutBackEvent
XCheckIfEvent: check the event	queue for a matching event	XCheckIfEvent
/return the next event in	queue matching type and window	XCheckTypedWindowEvent
don't/ /return the next event in	queue that matches event type;	XCheckTypedEvent
the request buffer (display all	queued requests) XFlush: flush	XFlush
stacking order XRaiseWindow:	raise a window to the top of the	XRaiseWindow
XDisplayKeycodes: obtain the	range of legal keycodes for a/	XDisplayKeycodes
XReadBitmapFile:	read a bitmap from disk	XReadBitmapFile
property XGetWMIconName:	read a window's XA_WM_ICON_NAME	XGetWMIconName
property XGetWMName:	read a window's XA_WM_NAME	XGetWMName
XGetWMNormalHints:	read a window's/	XGetWMNormalHints
property XGetWMSizeHints:	read a window's XA_WM_SIZE_HINTS	XGetWMSizeHints
XA_SIZE_HINTS XGetSizeHints:	read any property of type	XGetSizeHints

server/ XRefreshKeyboardMapping:	read keycode-keysym mapping from	XRefreshKeyboardMapping
properties XGetTextProperty:	read one of a window's text	XGetTextProperty
a zoomed window XGetZoomHints:	read the size hints property of	XGetZoomHints
property XGetWMHints:	read the window manager hints	XGetWMHints
XAllocNamedColor: allocate a	read-only colorcell from color/	XAllocNamedColor
closest/ XAllocColor: allocate a	read-only colormap cell with	XAllocColor
name /set RGB values of a	read/write colorcell by color	XStoreNamedColor
/set or change the RGB values of	read/write colorcells to the/	XStoreColors
/or change the RGB values of a	read/write colormap entry to the/	XStoreColor
XAllocColorPlanes: allocate	read/write (nonshareable) color/	XAllocColorPlanes
XAllocColorCells: allocate	read/write (nonshared)/	XAllocColorCells
client XRebindKeysym:	rebind a keysym to a string for	XRebindKeysym
that a top-level window be	reconfigured /request	XReconfigureWMWindow
draw an arc fitting inside a	rectangle XDrawArc:	XDrawArc
draw an outline of a	rectangle XDrawRectangle:	XDrawRectangle
XClipBox: generate the smallest	rectangle enclosing a region	XClipBox
XGetImage: place contents of a	rectangle from drawable into an/	XGetImage
location/ XGetSubImage: copy a	rectangle in drawable to a	XGetSubImage
XRectInRegion: determine if a	rectangle resides in a region	XRectInRegion
XUnionRectWithRegion: add a	rectangle to a region	XUnionRectWithRegion
draw the outlines of multiple	rectangles XDrawRectangles:	XDrawRectangles
a graphics context to a list of	rectangles /change clip_mask in	XSetClipRectangles
XFillRectangle: fill a	rectangular area	XFillRectangle
XClearArea: clear a	rectangular area in a window	XClearArea
XFillRectangles: fill multiple	rectangular areas	XFillRectangles
region XShrinkRegion:	reduce or expand the size of a	XShrinkRegion
smallest rectangle enclosing a	region XClipBox: generate the	XClipBox
create a new empty	region XCreateRegion:	XCreateRegion
storage associated with a	region /deallocate	XDestroyRegion
change offset of a	region XOffsetRegion:	XOffsetRegion
determine if a point is inside a	region XPointInRegion:	XPointInRegion
if a rectangle resides in a	region XRectInRegion: determine	XRectInRegion
context to the specified	region /of the graphics	XSetRegion
reduce or expand the size of a	region XShrinkRegion:	XShrinkRegion
add a rectangle to a	region XUnionRectWithRegion:	XUnionRectWithRegion
XSubtractRegion: subtract one	region from another	XSubtractRegion
XPolygonRegion: generate a	region from points	XPolygonRegion
XEmptyRegion: determine if a	region is empty	XEmptyRegion
compute the intersection of two	regions XIntersectRegion:	XIntersectRegion
compute the union of two	regions XUnionRegion:	XUnionRegion
union and intersection of two	regions /difference between the	XXorRegion
XEqualRegion: determine if two	regions have the same size,/	XEqualRegion
grab XUngrabButton:	release a button from a passive	XUngrabButton
XFreeCursor:	release a cursor	XFreeCursor
grab XUngrabKey:	release a key from a passive	XUngrabKey
active grab XUngrabKeyboard:	release the keyboard from an	XUngrabKeyboard
active grab XUngrabPointer:	release the pointer from an	XUngrabPointer
XUngrabServer:	release the server from grab	XUngrabServer
/destroy a client or its	remaining resources	XKillClient
control list XRemoveHost:	remove a host from the access	XRemoveHost
client's/ XChangeSaveSet: add or	remove a subwindow from the	XChangeSaveSet
client's/ XRemoveFromSaveSet:	remove a window from the	XRemoveFromSaveSet
access control/ XRemoveHosts:	remove multiple hosts from the	XRemoveHosts
both passed/ XCheckWindowEvent:	remove the next event matching	XCheckWindowEvent
matches mask XMaskEvent:	remove the next event that	XMaskEvent
matches mask;/ XCheckMaskEvent:	remove the next event that	XCheckMaskEvent
matches the/ XWindowEvent:	remove the next event that	XWindowEvent
XPeekEvent: get an event without	removing it from the queue	XPeekEvent

/by predicate procedure without	removing it from the queue	XPeekIfEvent
border pixel value attribute and	repaint the border /a window	XSetWindowBorder
window border tile attribute and	repaint the border /change a	XSetWindowBorderPixmap
connection to a/ XDisplayName:	report the display name (when	XDisplayName
number of/ XPending: flush the	request buffer and return the	XPending
events and/ XSync: flush the	request buffer and wait for all	XSync
queued/ XFlush: flush the	request buffer (display all	XFlush
be iconified XIconifyWindow:	request that a top-level window	XIconifyWindow
be/ XReconfigureWMWindow:	request that a top-level window	XReconfigureWMWindow
be withdrawn XWithdrawWindow:	request that a top-level window	XWithdrawWindow
list to allow or deny connection	requests /use access control	XEnableAccessControl
buffer (display all queued	requests) /flush the request	XFlush
XResetScreenSaver:	reset the screen saver	XResetScreenSaver
/determine if a rectangle	resides in a region	XRectInRegion
search prepared list for a given	resource XrmQGetSearchResource:	XrmQGetSearchResource
extract an option value from the	resource database XGetDefault:	XGetDefault
XrmDestroyDatabase: destroy a	resource database.	XrmDestroyDatabase
a resource specification to a	resource database /add	XrmPutLineResource
a resource specification into a	resource database /store	XrmPutResource
line/ XrmParseCommand: load a	resource database from command	XrmParseCommand
XrmPutFileDatabase: store a	resource database in a file	XrmPutFileDatabase
strings XrmGetResource: get a	resource from name and class as	XrmGetResource
the/ /obtain the GContext	(resource ID) associated with	XGContextFromGC
XrmInitialize: initialize the	resource manager	XrmInitialize
/to a database using a quark	resource name and string value	XrmQPutStringResource
/specification with separate	resource name and value	XrmPutStringResource
XrmQPutResource: store a	resource specification into a/	XrmQPutResource
XrmPutResource: store a	resource specification into a/	XrmPutResource
XrmQPutStringResource: add a	resource specification to a/	XrmQPutStringResource
XrmPutLineResource: add a	resource specification to a/	XrmPutLineResource
XrmPutStringResource: add a	resource specification with/	XrmPutStringResource
class as/ XrmQGetResource: get a	resource value using name and	XrmQGetResource
a client or its remaining	resources XKillClient: destroy	XKillClient
and pointer events when these	resources are grabbed /keyboard	XAllowEvents
XrmGetFileDatabase:	retrieve a database from a file	XrmGetFileDatabase
to X supported/ XListExtensions:	return a list of all extensions	XListExtensions
parent, and root XQueryTree:	return a list of children, ...	XQueryTree
XrmQGetSearchList:	return a list of database levels	XrmQGetSearchList
font names XListFonts:	return a list of the available	XListFonts
/copy a colormap and	return a new colormap ID	XCopyColormapAndFree
property name/ XInternAtom:	return an atom for a given	XInternAtom
XFetchBuffer:	return data from a cut buffer	XFetchBuffer
XFetchBytes:	return data from cut buffer 0	XFetchBytes
loaded font XQueryFont:	return information about a	XQueryFont
XGetKeyboardMapping:	return symbols for keycodes	XGetKeyboardMapping
focus window XGetInputFocus:	return the current keyboard	XGetInputFocus
XCheckTypedWindowEvent:	return the next event in queue/	XCheckTypedWindowEvent
that matches/ XCheckTypedEvent:	return the next event in queue	XCheckTypedEvent
/flush the request buffer and	return the number of pending/	XPending
XGetSelectionOwner:	return the owner of a selection	XGetSelectionOwner
XLookupColor: get database	RGB values and closest/	XLookupColor
XQueryColor: obtain the	RGB values and flags for a/	XQueryColor
colorcells XQueryColors: obtain	RGB values for an array of	XQueryColors
or/ XParseColor: look up	RGB values from ASCII color name	XParseColor
/and closest hardware-supported	RGB values from color name	XLookupColor
colorcell/ XStoreNamedColor: set	RGB values of a read/write	XStoreNamedColor
XStoreColor: set or change the	RGB values of a read/write/	XStoreColor
XStoreColors: set or change the	RGB values of read/write/	XStoreColors

structure XSetRGBColormaps:	set an XStandardColormap	XSetRGBColormaps
create a window and	set attributes XCreateWindow:	XCreateWindow
context to the/ XSetRegion:	set clip_mask of the graphics	XSetRegion
graphics context XSetClipMask:	set clip_mask pixmap in a	XSetClipMask
modifiers/ XSetModifierMapping:	set keycodes to be used as	XSetModifierMapping
manager /set the minimum	set of properties for the window	XSetStandardProperties
properties XSetTextProperty:	set one of a window's text	XSetTextProperty
a read/write/ XStoreColor:	set or change the RGB values of	XStoreColor
read/write/ XStoreColors:	set or change the RGB values of	XStoreColors
colorcell by/ XStoreNamedColor:	set RGB values of a read/write	XStoreNamedColor
context XSetArcMode:	set the arc mode in a graphics	XSetArcMode
attribute/ XSetWindowBackground:	set the background pixel value	XSetWindowBackground
in a graphics/ XSetBackground:	set the background pixel value	XSetBackground
operation in a/ XSetFunction:	set the bitwise logical	XSetFunction
graphics/ XSetClipOrigin:	set the clip origin in a	XSetClipOrigin
window XSetWindowColormap:	set the colormap attribute for a	XSetWindowColormap
graphics context XSetFont:	set the current font in a	XSetFont
context XSetFillRule:	set the fill rule in a graphics	XSetFillRule
context XSetFillStyle:	set the fill style in a graphics	XSetFillStyle
context XSetTile:	set the fill tile in a graphics	XSetTile
XSetFontPath:	set the font search path	XSetFontPath
logical function,/ XSetState:	set the foreground, background,	XSetState
in a graphics/ XSetForeground:	set the foreground pixel value	XSetForeground
XSetGraphicsExposures:	set the graphics_exposures/	XSetGraphicsExposures
XSetInputFocus:	set the keyboard focus window	XSetInputFocus
in a/ XSetLineAttributes:	set the line drawing components	XSetLineAttributes
XSetStandardProperties:	set the minimum set of/	XSetStandardProperties
a window's icon XSetIconName:	set the name to be displayed in	XSetIconName
XSetSelectionOwner:	set the owner of a selection	XSetSelectionOwner
saver XSetScreenSaver:	set the parameters of the screen	XSetScreenSaver
context XSetPlaneMask:	set the plane mask in a graphics	XSetPlaneMask
XSetPointerMapping:	set the pointer button mapping	XSetPointerMapping
window in/ XSetNormalHints:	set the size hints property of a	XSetNormalHints
zoomed window XSetZoomHints:	set the size hints property of a	XSetZoomHints
XStringListToTextProperty:	set the specified list of/	XStringListToTextProperty
context XSetStipple:	set the stipple in a graphics	XSetStipple
graphics/ XSetSubwindowMode:	set the subwindow mode in a	XSetSubwindowMode
graphics context XSetTSOrigin:	set the tile/stipple origin in a	XSetTSOrigin
type/ XSetSizeHints:	set the value of any property of	XSetSizeHints
XA_WM_ICON_SIZE/ XSetIconSizes:	set the value of the	XSetIconSizes
a window XSetClassHint:	set the XA_WM_CLASS property of	XSetClassHint
(command line/ XSetCommand:	set the XA_WM_COMMAND atom	XSetCommand
property/ XSetTransientForHint:	set the XA_WM_TRANSIENT_FOR	XSetTransientForHint
XChangeWindowAttributes:	set window attributes	XChangeWindowAttributes
have the same size, offset, and	shape /determine if two regions	XEqualRegion
the fastest supported stipple	shape XQueryBestStipple: obtain	XQueryBestStipple
the fastest supported fill tile	shape XQueryBestTile: obtain	XQueryBestTile
a mapping of modifier keys	(Shift, Control, etc.) /obtain	XGetModifierMapping
keycodes to be used as modifiers	(Shift, Control, etc.) /set	XSetModifierMapping
map a window on top of its	siblings XMapRaised:	XMapRaised
change the stacking order of	siblings XRestackWindows:	XRestackWindows
XGetPixel: obtain a	single pixel value from an image	XGetPixel
a drawable/ XCopyPlane: copy a	single plane of a drawable into	XCopyPlane
cursor, tile, or stipple	size /the "best" supported	XQueryBestSize
XResizeWindow: change a window's	size	XResizeWindow
XMoveResizeWindow: change the	size and position of a window	XMoveResizeWindow
/change the window position,	size, border width, or stacking/	XConfigureWindow
geometry/ /generate position and	size from standard window	XParseGeometry

XTextExtents: get	string and font metrics locally	XTextExtents
16-bit/ /query the server for	string and font metrics of a	XQueryTextExtents16
16-bit/ XTextExtents16: get	string and font metrics of a	XTextExtents16
/rebind a keysym to a	string for client	XRebindKeysym
XDrawString: draw an 8-bit text	string, foreground only	XDrawString
/map a key event to ASCII	string, keysym, and/	XLookupString
associated with the specified	string list /the in-memory data	XFreeStringList
metrics of a 16-bit character	string, locally /string and font	XTextExtents16
in pixels of an 8-bit character	string, locally /get the width	XTextWidth
in pixels of a 16-bit character	string, locally /get the width	XTextWidth16
its atom XGetAtomName: get a	string name for a property given	XGetAtomName
quark list /convert a key	string to a binding list and a	XrmStringToBindingQuarkList
/convert a keysym name	string to a keysym	XStringToKeysym
XrmStringToQuark: convert a	string to a quark	XrmStringToQuark
/convert a key	string to a quark list	XrmStringToQuarkList
using a quark resource name and	string value /to a database	XrmQPutStringResource
draw two-byte text	strings XDrawString16:	XDrawString16
XDrawText: draw 8-bit polytext	strings	XDrawText
draw 16-bit polytext	strings XDrawText16:	XDrawText16
resource from name and class as	strings XrmGetResource: get a	XrmGetResource
XTextProperty/ /obtain a list of	strings from a specified	XTextPropertyToStringList
/set the specified list of	strings to an XTextProperty/	XStringListToTextProperty
allocate an XClassHint	structure XAllocClassHint:	XAllocClassHint
allocate an XIconSize	structure XAllocIconSize:	XAllocIconSize
allocate an XSizeHints	structure XAllocSizeHints:	XAllocSizeHints
/allocate an XStandardColormap	structure	XAllocStandardColormap
allocate an XWMHints	structure XAllocWMHints:	XAllocWMHints
allocate memory for an XImage	structure XCreateImage:	XCreateImage
an entry from an XModifierKeymap	structure /delete	XDeleteModifiermapEntry
and free storage for the font	structure /unload a font	XFreeFont
free a keyboard modifier mapping	structure /destroy and	XFreeModifiermap
new entry to an XModifierKeymap	structure /add a	XInsertModifiermapEntry
load a font and fill information	structure XLoadQueryFont:	XLoadQueryFont
corresponding to a keycode in	structure /get the keysym	XLookupKeysym
a keyboard modifier mapping	structure /create	XNewModifiermap
set an XStandardColormap	structure XSetRGBColormaps:	XSetRGBColormaps
of strings to an XTextProperty	structure /the specified list	XStringListToTextProperty
from a specified XTextProperty	structure /a list of strings	XTextPropertyToStringList
/obtain the XStandardColormap	structure associated with the/	XGetRGBColormaps
/find the visual information	structures that match the/	XGetVisualInfo
XSetFillStyle: set the fill	style in a graphics context	XSetFillStyle
XSubImage: create a	subimage from part of an image	XSubImage
XSubtractRegion:	subtract one region from another	XSubtractRegion
XChangeSaveSet: add or remove a	subwindow from the client's/	XChangeSaveSet
XSetSubwindowMode: set the	subwindow mode in a graphics/	XSetSubwindowMode
and destroy a window and all	subwindows. /unmap	XDestroyWindow
XUnmapSubwindows: unmap all	subwindows of a given window	XUnmapSubwindows
XDestroySubwindows: destroy all	subwindows of a window	XDestroySubwindows
XMapSubwindows: map all	subwindows of window	XMapSubwindows
/change the keyboard preferences	such as key click	XChangeKeyboardControl
/a list of all extensions to X	supported by Xlib and the server	XListExtensions
/get the closest	supported cursor sizes	XQueryBestCursor
stipple/ /obtain the "best"	supported cursor, tile, or	XQueryBestSize
/obtain the fastest	supported fill tile shape	XQueryBestTile
XListPixmapFormats: obtain the	supported pixmap formats for a/	XListPixmapFormats
/obtain the fastest	supported stipple shape	XQueryBestStipple
/convert a keysym	symbol to a string	XKeysymToString
XGetKeyboardMapping: return	symbols for keycodes	XGetKeyboardMapping

XSynchronize: enable or disable	synchronization for debugging	XSynchronize
another /change the coordinate	system from one window to	XTranslateCoordinates
an entry from an association	table. XDeleteAssoc: delete	XDeleteAssoc
allocated for an association	table. /free the memory	XDestroyAssocTable
obtain data from an association	table XLookUpAssoc:	XLookUpAssoc
an entry in an association	table XMakeAssoc: create	XMakeAssoc
create a new association	table (X10) XCreateAssocTable:	XCreateAssocTable
that match the specified	template /information structures	XGetVisualInfo
/draw 8-bit image	text characters	XDrawImageString
/draw 16-bit image	text characters	XDrawImageString16
/read one of a window's	text properties	XGetTextProperty
/set one of a window's	text properties	XSetTextProperty
XDrawString: draw an 8-bit	text string, foreground only	XDrawString
XDrawString16: draw two-byte	text strings	XDrawString16
border /change a window border	tile attribute and repaint the	XSetWindowBorderPixmap
/change the background	tile attribute of a window	XSetWindowBackgroundPixmap
XSetTile: set the fill	tile in a graphics context	XSetTile
the "best" supported cursor,	tile, or stipple size /obtain	XQueryBestSize
the fastest supported fill	tile shape /obtain	XQueryBestTile
graphics/ XSetTSOrigin: set the	tile/stipple origin in a	XSetTSOrigin
stacking order /circulate the	top child to the bottom of the	XCirculateSubwindowsUp
XMapRaised: map a window on	top of its siblings	XMapRaised
/the bottom child to the	top of the stacking order	XCirculateSubwindowsDown
/raise a window to the	top of the stacking order	XRaiseWindow
XIconifyWindow: request that a	top-level window be iconified	XIconifyWindow
/request that a	top-level window be reconfigured	XReconfigureWMWindow
XWithdrawWindow: request that a	top-level window be withdrawn	XWithdrawWindow
values from ASCII color name or	translate hexadecimal value /RGB	XParseColor
auto-repeat/ XAutoRepeatOff:	turn off the keyboard	XAutoRepeatOff
keys XAutoRepeatOn:	turn on the keyboard auto-repeat	XAutoRepeatOn
XForceScreenSaver:	turn the screen saver on or off	XForceScreenSaver
/create a cursor from	two bitmaps	XCreatePixmapCursor
XDrawLine: draw a line between	two points	XDrawLine
compute the intersection of	two regions XIntersectRegion:	XIntersectRegion
compute the union of	two regions XUnionRegion:	XUnionRegion
the union and intersection of	two regions /difference between	XXorRegion
XEqualRegion: determine if	two regions have the same size,/	XEqualRegion
XDrawString16: draw	two-byte text strings	XDrawString16
entry for a given window and	type /delete a context	XDeleteContext
window /obtain the atom	type and property format for a	XGetWindowProperty
the next event in queue matching	type and window /return	XCheckTypedWindowEvent
in queue that matches event	type; don't wait /the next event	XCheckTypedEvent
/to a window and context	type (not graphics context)	XSaveContext
get the next event of any	type or window XNextEvent:	XNextEvent
/read any property of	type XA_SIZE_HINTS	XGetSizeHints
/set the value of any property of	type XA_SIZE_HINTS	XSetSizeHints
XSelectInput: select the event	types to be sent to a window	XSelectInput
default if/ XUninstallColormap:	uninstall a colormap; install	XUninstallColormap
/the difference between the	union and intersection of two/	XXorRegion
XUnionRegion: compute the	union of two regions	XUnionRegion
XUnloadFont:	unload a font.	XUnloadFont
for the font/ XFreeFont:	unload a font and free storage	XFreeFont
XUnmapWindow:	unmap a window	XUnmapWindow
window XUnmapSubwindows:	unmap all subwindows of a given	XUnmapSubwindows
all subwindows. XDestroyWindow:	unmap and destroy a window and	XDestroyWindow
XCreateSimpleWindow: create an	unmapped InputOutput window	XCreateSimpleWindow
/calculate window geometry given	user geometry string and default/	XGeometry
/specification to a database	using a quark resource name and/	XrmQPutStringResource

/get a resource value	using name and class as quarks	XrmQGetResource
specification into a database	using quarks /store a resource	XrmQPutResource
name or translate hexadecimal	value /values from ASCII color	XParseColor
with separate resource name and	value /a resource specification	XrmPutStringResource
a quark resource name and string	value /to a database using	XrmQPutStringResource
/change a window border pixel	value attribute and repaint the/	XSetWindowBorder
/set the background pixel	value attribute of a window	XSetWindowBackground
and/ XSaveContext: save a data	value corresponding to a window	XSaveContext
XGetPixel: obtain a single pixel	value from an image	XGetPixel
XGetDefault: extract an option	value from the resource database	XGetDefault
/set the background pixel	value in a graphics context	XSetBackground
/set the foreground pixel	value in a graphics context	XSetForeground
a constant value to every pixel	value in an image /add	XAddPixel
XPutPixel: set a pixel	value in an image	XPutPixel
XConvertSelection: use the	value of a selection	XConvertSelection
XSetSizeHints: set the	value of any property of type/	XSetSizeHints
property XSetIconSizes: set the	value of the XA_WM_ICON_SIZE	XSetIconSizes
image XAddPixel: add a constant	value to every pixel value in an	XAddPixel
XrmQGetResource: get a resource	value using name and class as/	XrmQGetResource
with depth, applying pixel	values /drawable into a drawable	XCopyPlane
XLookupColor: get database RGB	values and closest/	XLookupColor
XQueryColor: obtain the RGB	values and flags for a specified/	XQueryColor
XQueryColors: obtain RGB	values for an array of/	XQueryColors
XParseColor: look up RGB	values from ASCII color name or/	XParseColor
closest hardware-supported RGB	values from color name /and	XLookupColor
by/ XStoreNamedColor: set RGB	values of a read/write colorcell	XStoreNamedColor
entry to/ /set or change the RGB	values of a read/write colormap	XStoreColor
to the/ /set or change the RGB	values of read/write colorcells	XStoreColors
the/ XQueryKeymap: obtain a bit	vector for the current state of	XQueryKeymap
draw a polyline or curve between	vertex list (from X10) XDraw:	XDraw
a filled polygon or curve from	vertex list (from X10) /draw	XDrawFilled
obtain the visual ID from a	Visual XVisualIDFromVisual:	XVisualIDFromVisual
XVisualIDFromVisual: obtain the	visual ID from a Visual	XVisualIDFromVisual
that/ XGetVisualInfo: find the	visual information structures	XGetVisualInfo
XMatchVisualInfo: obtain the	visual information that matches/	XMatchVisualInfo
event that matches mask; don't	wait /remove the next	XCheckMaskEvent
that matches event type; don't	wait /the next event in queue	XCheckTypedEvent
window and passed mask; don't	wait /event matching both passed	XCheckWindowEvent
to/ /flush the request buffer and	wait for all events and errors	XSync
predicate procedure XIfEvent:	wait for event matched in	XIfEvent
fails) /report the display name	(when connection to a display	XDisplayName
character/ XTextWidth16: get the	width in pixels of a 16-bit	XTextWidth16
character/ XTextWidth: get the	width in pixels of an 8-bit	XTextWidth
/change the border	width of a window	XSetWindowBorderWidth
window position, size, border	width, or stacking order /the	XConfigureWindow
a property associated with a	window XChangeProperty: change	XChangeProperty
event in queue matching type and	window /return the next	XCheckTypedWindowEvent
clear a rectangular area in a	window XClearArea:	XClearArea
XClearWindow: clear an entire	window	XClearWindow
create an unmapped InputOutput	window XCreateSimpleWindow:	XCreateSimpleWindow
assign a cursor to a	window XDefineCursor:	XDefineCursor
destroy all subwindows of a	window XDestroySubwindows:	XDestroySubwindows
the XA_WM_CLASS property of a	window XGetClassHint: get	XGetClassHint
the current keyboard focus	window XGetInputFocus: return	XGetInputFocus
property of a	window /the XA_WM_TRANSIENT_FOR	XGetTransientForHint
obtain the current attributes of	window XGetWindowAttributes:	XGetWindowAttributes
type and property format for a	window /obtain the atom	XGetWindowProperty
size hints property of a zoomed	window XGetZoomHints: read the	XGetZoomHints

Introduction

This page describes the format of each reference page in this volume.

Name

XFunctionName — brief description of the function.

Synopsis

The Synopsis section presents the calling syntax for the routine, including the declarations of the arguments and return type. For example:

```
returntype XFunctionName(arg1, arg2, arg3);
      type1 arg1;
      type2 *arg2;            /* RETURN */
      type3 *arg3;            /* SEND and RETURN */
```

The return type `Status` is of type `int`; it returns either `True` or `False` to indicate whether the routine was successful.

Arguments

The Arguments section describes each of the arguments used by the function. There are three sorts of arguments: arguments that specify data to the function, arguments that return data from the function, and arguments that do both. An example of each type is shown below:

arg1	Specifies information for `XFunctionName`. The description of arguments that pass data to the function always begins with the word "Specifies," as shown in this example.
arg2	Returns a pointer to data to be filled in by `XFunctionName`. The description of arguments that return data from the function always begins with the word "Returns."
arg3	Specifies information for `XFunctionName`, and returns data from the function. The description of arguments that both pass data to the function and return data from the function uses both the words "Specifies" and "Returns."

Availability

The Availability section specifies that a given function is only available in Release 4 and later releases. If there is no Availability section, the function is available prior to Release 4.

Description

The Description section describes 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 pointers to related information in both volumes of this manual.

Structures

The Structures section contains the C definitions of the X-specific data types used by `FunctionName` as arguments or return values. It also contains definitions of important constants used by the function. Additional structures not shown can be found in Appendix F, *Structure Reference*.

Errors

The general description of the error types is contained in Appendix B, *Error Messages and Protocol Requests*. Some functions generate errors due to function-specific interpretation of arguments. Where appropriate, these function-specific causes have been listed along with the error event types they generate.

Related Commands

The Related Commands section lists the Xlib functions and macros related to `XFunctionName`.

XActivateScreenSaver

Name

XActivateScreenSaver — activate screen blanking.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.

Description

`XActivateScreenSaver` turns on the screen saver using the parameters set with `XSet-ScreenSaver`. The screen saver blanks the screen or makes random changes to the display in order to save the phosphors from burnout when the screen is left unattended for an extended period of time. The interval that the server will wait before starting screen save activity can be set with `XSetScreenSaver`. Exactly how the screen saver works is server-dependent.

For more information on the screen saver, see Volume One, Chapter 13, *Other Programming Techniques*.

Related Commands

`XForceScreenSaver`, `XGetScreenSaver`, `XResetScreenSaver`, `XSetScreen-Saver`.

Name

XAddHost — add a host to the access control list.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 server specified by *display*. The access control list is a primitive security feature that allows access to the server only by other machines listed in a file on the machine running the server. On UNIX-based systems, this file is called */etc/X?.hosts*, where *?* is the number of the server.

The application that calls `XAddHost` and the server whose list is being updated must be running on the same host machine.

The `address` data must be a valid address for the type of network in which the server operates, as specified in the `family` member. Internet, DECnet and ChaosNet networks are currently supported.

For TCP/IP, the address should be in network byte order. For the DECnet family, the server performs no automatic swapping on the address bytes. A Phase IV address is two bytes long. The first byte contains the least significant eight bits of the node number. The second byte contains the most significant two bits of the node number in the least significant two bits of the byte, and the area in the most significant six bits of the byte.

For more information on access control, see Volume One, Chapter 13, *Other Programming Techniques*.

Structures

```
typedef struct {
    int family;              /* for example FamilyInternet */
    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

```
BadAccess
BadValue
```

Related Commands

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

XAddHosts

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 connection to an X server; returned from `XOpenDisplay`.
hosts	Specifies each host that is to be added.
num_hosts	Specifies the number of hosts that are to be added.

Description

`XAddHosts` adds each specified host to the access control list for the server specified by *display*. The access control list is a primitive security feature that allows access to the server only by other machines listed in a file on the machine running the server. On UNIX systems, this file is */etc/X?.hosts*, where *?* is the number of the display.

The application that calls `XAddHosta` and the server whose list is being updated must be running on the same host machine.

The `address` data must be a valid address for the type of network in which the server operates, as specified by the `family` member. Internet, DECnet and ChaosNet networks are currently supported.

For TCP/IP, the address should be in network byte order. For the DECnet family, the server performs no automatic swapping on the address bytes. A Phase IV address is two bytes long. The first byte contains the least significant eight bits of the node number. The second byte contains the most significant two bits of the node number in the least significant two bits of the byte, and the area in the most significant six bits of the byte.

For more information on access control, see Volume One, Chapter 13, *Other Programming Techniques*.

Structures

```
typedef struct {
    int family;              /* for example Family Internet */
    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

BadAccess
BadValue

Related Commands

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

Name

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

Synopsis

```
XAddPixel(ximage, value)
      XImage *ximage;
      unsigned long value;
```

Arguments

ximage	Specifies a pointer to the image to be modified.
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 causes an overflow, 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.

For more information on images, see Volume One, Chapter 6, *Drawing Graphics and Text*.

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;                 /* quantity of scan line 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;              /* accelerator 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 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;
```

Related Commands

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

XAddToSaveSet

Name

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

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window you want to add to the client's save-set.

Description

`XAddToSaveSet` adds the specified window to the 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 titlebar or other decorations for each application. When the window manager dies unexpectedly, the windows in the save-set are reparented to their closest living ancestor, so that they remain alive. See Volume One, Chapter 13, *Other Programming Techniques*, for more information about save-sets.

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

Errors

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

Related Commands

`XChangeSaveSet`, `XRemoveFromSaveSet`.

XAllocClassHint

Name

XAllocClassHint — allocate an `XClassHint` structure.

Synopsis

```
XClassHint *XAllocClassHint()
```

Availability

Release 4 and later.

Description

`XAllocClassHint` allocates and returns a pointer to an `XClassHint` structure, for use in calling `XSetWMProperties`, `XGetClassHint`, or `XSetClassHint`. Note that the pointer fields in the `XClassHint` structure are initially set to `NULL`. If insufficient memory is available, `XAllocClassHint` returns `NULL`. To free the memory allocated to this structure, use `XFree`.

The purpose of this function is to avoid compiled-in structure sizes, so that object files will be binary compatible with later releases that may have new members added to structures.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    char *res_name;
    char *res_class;
} XClassHint;
```

Related Commands

`XGetClassHint`, `XSetClassHint`, `XSetWMProperties`.

XAllocColor

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; /* SENDs and RETURNs */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the ID of the colormap in which the colorcell is to be allocated.
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 cells. 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, `XAllocColor` returns a `Status` of zero (in read/write colormaps, it does not return the closest available read-only colorcell that has already been allocated). If it succeeds, `XAllocColor` returns nonzero.

Note that *colorcell_def* stores both the requested color when `XAllocColor` is called and the result when `XAllocColor` returns.

`XAllocColor` does not use or affect the `flags` member of the `XColor` structure.

For more information, see Volume One, Chapter 7, *Color*.

Structures

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

Errors

`BadColormap`

Related Commands

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

XAllocColorCells

Name

XAllocColorCells — allocate read/write (nonshared) 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 connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the ID of the colormap in which the colorcell is to be allocated.
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 nonnegative.
pixels	Returns an array of pixel values.
ncolors	Specifies the number of pixel values returned in the *pixels* array. Must be positive.

Description

`XAllocColorCells` 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`, `XStoreColors`, or `XStoreNamedColor`.

`Status` is zero on failure, and nonzero on success.

For more information, see Volume One, Chapter 7, *Color*.

Errors

BadColormap	
BadValue	*nplanes* is negative. *ncolors* is not positive.

Related Commands

BlackPixel, WhitePixel, XAllocColor, XAllocColorPlanes, XAllocNamed-Color, XFreeColors, XLookupColor, XParseColor, XQueryColor, XQuery-Colors, XStoreColor, XStoreColors, XStoreNamedColor.

Name

XAllocColorPlanes — allocate read/write (nonshareable) 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 connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the ID of the colormap to be used.
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 planes (shades). Must be nonnegative.
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 to 1 respectively. Unique pixel values are generated by by ORing together subsets of masks with each item in the *pixels* list (*pixels* does not by itself contain pixel values). In doing this, note that *ncolors*$*(2^{(nreds+ngreens+nblues)})$ distinct pixel values are allocated.

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.

Note, however, that there are actually only *ncolors*$*(2^{nreds})$ independent red entries, *ncolors*^$*(2^{ngreens})$ independent green entries, and *ncolors*$*(2^{nblues})$ independent blue entries in the colormap. This is true even for `PseudoColor`. This does not cause problems, though, because when the colormap entry for a pixel value is changed using `XStoreColors` or `XStoreNamedColor`, the pixel is decomposed according to *rmask*, *gmask*, and *bmask* and the corresponding pixel subfield entries are updated.

`Status` is zero on failure, and nonzero on success.

For more information, see Volume One, Chapter 7, *Color*.

Errors

`BadColormap`	
`BadValue`	*ncolors* is not positive. At least one of *nreds*, *ngreens*, *nblues* is negative.

Related Commands

`BlackPixel`, `WhitePixel`, `XAllocColor`, `XAllocColorCells`, `XAllocNamed-Color`, `XFreeColors`, `XLookupColor`, `XParseColor`, `XQueryColor`, `XQuery-Colors`, `XStoreColor`, `XStoreColors`, `XStoreNamedColor`.

XAllocIconSize

Name

XAllocIconSize — allocate an `XIconSize` structure.

Synopsis

```
XIconSize *XAllocIconSize()
```

Availability

Release 4 and later.

Description

`XAllocIconSize` allocates and returns a pointer to an `XIconSize` structure, for use in calling `XGetIconSizes` or `XSetIconSizes`. Note that all fields in the `XIconSize` structure are initially set to zero. If insufficient memory is available, `XAllocIconSize` returns `NULL`. To free the memory allocated to this structure, use `XFree`.

The purpose of this function is to avoid compiled-in structure sizes, so that object files will be binary compatible with later releases that may have new members added to structures.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    int min_width, min_height;
    int max_width, max_height;
    int width_inc, height_inc;
} XIconSize;
```

Related Commands

`XGetIconSizes`, `XSetIconSizes`.

XAllocNamedColor

Name

XAllocNamedColor — allocate a 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 connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the ID of the colormap in which the colorcell will be allocated.
colorname	Specifies the color name string (for example, "red") you want. Upper or lower case does not matter. The string should be in ISO LATIN-1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.
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 colorcell with the closest color available, as described under `XAllocColor`. Both the 'exact' database 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, and is a `StaticColor`, `DirectColor`, or `StaticGray` visual class, `XAllocNamedColor` returns the closest read-only colorcell already allocated, and does not actually create or set any new colorcell. If the colormap is full and is a `PseudoColor`, `TrueColor`, or `GrayScale` visual class, `XAllocNamedColor` fails and returns zero.

`XAllocNamedColor` returns a `Status` of zero if *colorname* was not found in the database or if the color could not be allocated. The function returns nonzero when it succeeds.

For more information, see Volume One, Chapter 7, *Color*.

Errors

```
BadColormap
BadName
```

Structures

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

Related Commands

BlackPixel, WhitePixel, XAllocColor, XAllocColorCells, XAllocColor-Planes, XFreeColors, XLookupColor, XParseColor, XQueryColor, XQuery-Colors, XStoreColor, XStoreColors, XStoreNamedColor.

XAllocSizeHints

Name

XAllocSizeHints — allocate an `XSizeHints` structure.

Synopsis

```
XSizeHints *XAllocSizeHints()
```

Availability

Release 4 and later.

Description

`XAllocSizeHints` allocates and returns a pointer to an `XSizeHints` structure, for use in calling `XSetWMProperties`, `XSetWMNormalHints`, or `XGetWMNormalHints`. Note that all fields in the `XSizeHints` structure are initially set to zero. If insufficient memory is available, `XAllocSizeHints` returns `NULL`. To free the memory allocated to this structure, use `XFree`.

The purpose of this function is to avoid compiled-in structure sizes, so that object files will be binary compatible with later releases that may have new members added to structures.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    long flags;      /* marks which fields in this structure are defined */
    int x, y;        /* Obsolete */
    int width, height;  /* Obsolete */
    int min_width, min_height;
    int max_width, max_height;
    int width_inc, height_inc;
    struct {
        int x;       /* numerator */
        int y;       /* denominator */
    } min_aspect, max_aspect;
    int base_width, base_height;
    int win_gravity;
} XSizeHints;
```

Related Commands

`XGetWMNormalHints`, `XSetWMNormalHints`, `XSetWMProperties`.

XAllocStandardColormap

Name

XAllocStandardColormap — allocate an `XStandardColormap` structure.

Synopsis

```
XStandardColormap *XAllocStandardColormap()
```

Availability

Release 4 and later.

Description

`XAllocStandardColormap` allocates and returns a pointer to an `XStandardColormap` structure for use in calling `XGetRGBColormaps` or `XSetRGBColormaps`. Note that all fields in the `XStandardColormap` structure are initially set to zero. If insufficient memory is available, `XAllocStandardColormap` returns `NULL`. To free the memory allocated to this structure, use `XFree`.

The purpose of this function is to avoid compiled-in structure sizes, so that object files will be binary compatible with later releases that may have new members added to structures.

For more information, see Volume One, Chapter 7, *Color*.

Structures

```
/* value for killid field */

#define   ReleaseByFreeingColormap      ( (XID) 1L)

typedef struct {
     Colormap colormap;
     unsigned long red_max;
     unsigned long red_mult;
     unsigned long green_max;
     unsigned long green_mult;
     unsigned long blue_max;
     unsigned long blue_mult;
     unsigned long base_pixel;
     VisualID visualid;
     XID killid;
} XStandardColormap;
```

Related Commands

`XGetRGBColormaps`, `XSetRGBColormaps`.

XAllocWMHints

Name

XAllocWMHints — allocate an `XWMHints` structure.

Synopsis

```
XWMHints *XAllocWMHints()
```

Availability

Release 4 and later.

Description

The `XAllocWMHints` function allocates and returns a pointer to an `XWMHints` structure, for use in calling `XSetWMProperties`, `XSetWMHints`, or `XGetWMHints`. Note that all fields in the `XWMHints` structure are initially set to zero. If insufficient memory is available, `XAllocWMHints` returns `NULL`. To free the memory allocated to this structure, use `XFree`.

The purpose of this function is to avoid compiled-in structure sizes, so that object files will be binary compatible with later releases that may have new members added to structures.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    long flags;           /* marks which fields in this structure are defined */
    Bool input;           /* does this application rely on the window manager
                                to get keyboard input? */
    int initial_state;    /* see below */
    Pixmap icon_pixmap;   /* pixmap to be used as icon */
    Window icon_window;   /* window to be used as icon */
    int icon_x, icon_y;   /* initial position of icon */
    Pixmap icon_mask;     /* pixmap to be used as mask for icon_pixmap */
    XID window_group;     /* id of related window group */
    /* this structure may be extended in the future */
} XWMHints;
```

Related Commands

`XGetWMHints`, `XSetWMHints`, `XSetWMProperties`.

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 connection to an X server; 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 (not released to Xlib to propagate into Xlib's queues) 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, 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 occurred 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 occurred 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 will be released.

For more information on event handling, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Errors

`BadValue`	Invalid mode constant.

Related Commands

`QLength`, `XCheckIfEvent`, `XCheckMaskEvent`, `XCheckTypedEvent`, `XCheck-TypedWindowEvent`, `XCheckWindowEvent`, `XEventsQueued`, `XGetInputFocus`, `XGetMotionEvents`, `XIfEvent`, `XMaskEvent`, `XNextEvent`, `XPeekEvent`, `XPeek-IfEvent`, `XPending`, `XPutBackEvent`, `XSelectInput`, `XSendEvent`, `XSetInput-Focus`, `XSynchronize`, `XWindowEvent`.

Name

XAutoRepeatOff — turn off the keyboard auto-repeat keys.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.

Description

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

Related Commands

`XAutoRepeatOn`, `XBell`, `XChangeKeyboardControl`, `XGetDefault`, `XGet-KeyboardControl`, `XGetPointerControl`.

XAutoRepeatOn

Name

XAutoRepeatOn — turn on the keyboard auto-repeat keys.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.

Description

`XAutoRepeatOn` sets the keyboard to auto-repeat; that is, holding any non-modal key down will result in multiple `KeyPress` and `KeyRelease` event pairs with the same `keycode` member. Keys such as Shift Lock will still not repeat.

Related Commands

`XAutoRepeatOff`, `XBell`, `XChangeKeyboardControl`, `XGetDefault`, `XGet-KeyboardControl`, `XGetPointerControl`.

Name

XBell — ring the bell (Control G).

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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, 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 nonnegative 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 `XChange-KeyboardControl`.

Errors

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

Related Commands

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

XChangeActivePointerGrab

Name

XChangeActivePointerGrab — change the parameters of an active pointer grab.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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`, `Button1-MotionMask`, `Button2MotionMask`, `Button3MotionMask`, `But-ton4MotionMask`, `Button5MotionMask`, `ButtonMotionMask`, `KeymapStateMask`.
cursor	Specifies the cursor that is displayed. A value of `None` 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 characteristics of an active pointer grab, if 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 `XGrab-Button`, or the automatic grab that occurs between `ButtonPress` and `ButtonRelease`.

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

For more information on pointer grabbing, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Errors

`BadCursor`	
`BadValue`	The *event_mask* argument is invalid.

Related Commands

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

Name

XChangeGC — change the components of a given graphics context.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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; it is made by combining any number of the mask symbols listed in the Structures section using bitwise OR (|). 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 performance 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.

For more information, see Volume One, Chapter 5, *The Graphics Context*, and Chapter 6, *Drawing Graphics and Text*.

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

Related Commands

DefaultGC, XCopyGC, XCreateGC, XFreeGC, XGContextFromGC, XGetGCValues, XSetArcMode, XSetBackground, XSetClipMask, XSetClipOrigin, XSetClip-Rectangles, XSetDashes, XSetFillRule, XSetFillStyle, XSetForeground, XSetFunction, XSetGraphicsExposures, XSetLineAttributes, XSetPlane-Mask, XSetRegion, XSetState, XSetStipple, XSetSubwindowMode, XSet-TSOrigin.

XChangeKeyboardControl

Name

XChangeKeyboardControl — change the keyboard preferences such as key click.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
value_mask	Specifies a mask composed of ORed symbols from the table shown in the Structures section below, 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, light state, and keyboard auto-repeat. Changing some or all these settings may not be possible on all servers.

The *value_mask* argument specifies which values are to be changed; it is made by combining any number of the mask symbols listed in the Structures section using bitwise OR (|).

The *values* structure contains the values to be set, as follows:

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

`bell_percent` sets the base volume for the bell between 0 (off) and 100 (loud) inclusive. Setting to –1 restores the default.

`bell_pitch` sets the pitch (specified in Hz) of the bell. Setting to –1 restores the default.

`bell_duration` sets the duration (specified in milliseconds) of the bell. Setting to -1 restores the default.

`led_mode` is either `LedModeOn` or `LedModeOff`. `led` is a number between 1 and 32 inclusive that specifies which light's state is to be changed. If both `led_mode` and `led` are specified, then the state of the LED specified in `led` is changed to the state specified in `led_mode`. If only `led_mode` is specified, then all the LEDs assume the value specified by `led_mode`.

`auto_repeat_mode` is either `AutoRepeatModeOn`, `AutoRepeatModeOff`, or `AutoRepeatModeDefault`. `key` is a keycode between 7 and 255 inclusive. If both `auto_repeat_mode` and `key` are specified, then the auto-repeat mode of the key specified by `key` is set as specified by `auto_repeat_mode`. If only `auto_repeat_mode` is specified, then the global auto repeat mode for the entire keyboard is changed, without affecting the settings for each key. If the `auto_repeat_mode` is `AutoRepeatModeDefault` for either case, the key or the entire keyboard is returned to its default setting for the server, which is normally to have all non-modal keys repeat.

When a key is being used as a modifier key, it does not repeat regardless of the individual or global auto repeat mode.

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

For more information on user preferences, see Volume One, Chapter 9, *The Keyboard and Pointer*.

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;              /* LedModeOn or LedModeOff */
     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`.

Related Commands

`XAutoRepeatOff`, `XAutoRepeatOn`, `XBell`, `XGetDefault`, `XGetKeyboard-Control`, `XGetPointerControl`.

XChangeKeyboardMapping

Name

XChangeKeyboardMapping — change the 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 connection to an X server; 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 keysyms for the specified number of keycodes. The symbols for keycodes outside this range remain 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` field of the `Display` structure (else a `BadValue` error):

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

The keysym number *N* (counting from 0) for keycode *K* has an index in the *keysyms* array (counting from 0) 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, sent to this and all other clients, since the keycode to keysym mapping is global to all clients.

Errors

`BadAlloc`	
`BadValue`	*`first.keycode`* less than *`display`*->`min_keycode`. *`display`*->`max_keycode` exceeded (see above).

Related Commands

`XDeleteModifiermapEntry`, `XFreeModifiermap`, `XGetKeyboardMapping`, `XGetModifierMapping`, `XInsertModifiermapEntry`, `XKeycodeToKeysym`, `XKeysymToKeycode`, `XKeysymToString`, `XLookupKeysym`, `XLookupString`, `XNewModifierMap`, `XQueryKeymap`, `XRebindKeySym`, `XRefreshKeyboard-Mapping`, `XSetModifierMapping`, `XStringToKeysym`.

XChangePointerControl

Name

XChangePointerControl — change the pointer preferences.

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 connection to an X server; 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 functions. The acceleration is a fraction (*accel_numerator/accel_denominator*) which specifies how many times faster than normal the sprite on the screen moves for a given pointer movement. Acceleration takes effect 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 nonzero for the pointer values to be set; otherwise, the parameters will be unchanged. Setting any of the last three arguments to –1 restores the default for that argument.

The fraction may be rounded arbitrarily by the server.

Errors

`BadValue`	*accel_denominator* is 0. Negative value for *do_accel* or *do_threshold*.

Related Commands

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

XChangeProperty

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 connection to an X server; returned from `XOpenDisplay`.
w	Specifies the 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 `XChange-Property`. Possible values are `8`, `16`, and `32`.
mode	Specifies the mode of the operation. Possible values are `PropMode-Replace`, `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 and stores the new data.

- `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 insufficient memory.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadAlloc`
`BadAtom`
`BadMatch`
`BadValue`
`BadWindow`

Related Commands

`XDeleteProperty`, `XGetAtomName`, `XGetFontProperty`, `XGetWindowProperty`, `XInternAtom`, `XListProperties`, `XRotateWindowProperties`, `XSetStandard-Properties`.

XChangeSaveSet

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 connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of 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` adds or deletes windows from a client's save-set. This client is usually the window manager.

The save-set of the window manager is a list of other client's top-level windows which have been reparented. If the window manager dies unexpectedly, these top-level application windows are children of a window manager window and therefore would normally be destroyed. The save-set prevents this by automatically reparenting the windows listed in the save-set to their closest existing ancestor, and then remapping them.

Windows are removed automatically from the save-set by the server when they are destroyed.

For more information on save-sets, see Volume One, Chapter 13, *Other Programming Techniques*.

Errors

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

Related Commands

`XAddToSaveSet`, `XRemoveFromSaveSet`.

XChangeWindowAttributes

Name

XChangeWindowAttributes — set window attributes.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window ID.
valuemask	Specifies which window attributes are defined in the *attributes* argument. The mask is made by combining the appropriate mask symbols listed in the Structures section using bitwise OR (|). If *valuemask* is zero, the rest is ignored, and *attributes* is not referenced. The values and restrictions are the same as for `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, see Volume One, Chapter 4, *Window Attributes*.

Changing the background does not cause the window contents to be changed immediately–not until the next `Expose` event or `XClearWindow` call. Drawing into the pixmap that was set as the background pixmap attribute has an undefined effect on the window background. The server may or may not make a copy of the pixmap. Setting the border causes the border to be repainted immediately. 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` attributes 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`, `Resize-`

RedirectMask, 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 colormap may not be installed until the window manager calls XInstall-Colormap).

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

For more information, see Volume One, Chapter 2, *X Concepts*, and Chapter 4, *Window Attributes*.

Structures

```
/*
 * Data structure for setting window attributes.
 */
typedef struct {
    Pixmap background_pixmap;          /* pixmap, None, or ParentRelative */
    unsigned long background_pixel;    /* background pixel */
    Pixmap border_pixmap;              /* pixmap, None, or CopyFromParent */
    unsigned long border_pixel;        /* border pixel value */
    int bit_gravity;                   /* one of bit gravity values */
    int win_gravity;                   /* one of the window gravity values */
    int backing_store;                 /* NotUseful, WhenMapped, Always */
    unsigned long backing_planes;      /* planes to be 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;

/* Definitions for valuemask argument of CreateWindow and ChangeWindowAttributes */

#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`
`BadColormap`
`BadCursor`
`BadMatch`
`BadPixmap`
`BadValue`
`BadWindow`

Related Commands

`XGetGeometry`, `XGetWindowAttributes`, `XSetWindowBackground`, `XSet-WindowBackgroundPixmap`, `XSetWindowBorder`, `XSetWindowBorderPixmap`.

XCheckIfEvent

Name

XCheckIfEvent — check the event queue for a matching event.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
event	Returns the matched event.
predicate	Specifies the procedure that is called to determine if the next event matches your criteria.
arg	Specifies the user-specified argument 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. If found, that event is removed from the queue, and `True` is returned. If no match is found, `XCheckIfEvent` returns `False` and flushes the request buffer. No other events are removed from the queue. Later events in the queue are not searched.

The predicate procedure is called with the arguments *display*, *event*, and *arg*.

For more information, see Volume One, Chapter 8, *Events*.

Related Commands

`QLength`, `XAllowEvents`, `XCheckMaskEvent`, `XCheckTypedEvent`, `XCheck-TypedWindowEvent`, `XCheckWindowEvent`, `XEventsQueued`, `XGetInputFocus`, `XGetMotionEvents`, `XIfEvent`, `XMaskEvent`, `XNextEvent`, `XPeekEvent`, `XPeek-IfEvent`, `XPending`, `XPutBackEvent`, `XSelectInput`, `XSendEvent`, `XSetInput-Focus`, `XSynchronize`, `XWindowEvent`.

XCheckMaskEvent

Name

XCheckMaskEvent — remove the next event that matches mask; don't wait.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 that 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 request buffer and immediately returns `False`, without waiting.

For more information, see Volume One, Chapter 8, *Events*.

Related Commands

`QLength`, `XAllowEvents`, `XCheckIfEvent`, `XCheckTypedEvent`, `XCheckTyped-WindowEvent`, `XCheckWindowEvent`, `XEventsQueued`, `XGetInputFocus`, `XGet-MotionEvents`, `XIfEvent`, `XMaskEvent`, `XNextEvent`, `XPeekEvent`, `XPeek-IfEvent`, `XPending`, `XPutBackEvent`, `XSelectInput`, `XSendEvent`, `XSetInput-Focus`, `XSynchronize`, `XWindowEvent`.

XCheckTypedEvent

Name

XCheckTypedEvent — return the next event in queue that matches event type; don't wait.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 request 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 more information, see Volume One, Chapter 8, *Events*.

Related Commands

`QLength`, `XAllowEvents`, `XCheckIfEvent`, `XCheckMaskEvent`, `XCheckTyped-WindowEvent`, `XCheckWindowEvent`, `XEventsQueued`, `XGetInputFocus`, `XGet-MotionEvents`, `XIfEvent`, `XMaskEvent`, `XNextEvent`, `XPeekEvent`, `XPeek-IfEvent`, `XPending`, `XPutBackEvent`, `XSelectInput`, `XSendEvent`, `XSetInput-Focus`, `XSynchronize`, `XWindowEvent`.

XCheckTypedWindowEvent

Name

XCheckTypedWindowEvent — return the 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 connection to an X server; 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 request buffer and returns `False` if the event is not found.

For more information, see Volume One, Chapter 8, *Events*.

Related Commands

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

XCheckWindowEvent

Name

XCheckWindowEvent — remove the next event matching both passed window and passed mask; don't wait.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 that 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.

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

For more information, see Volume One, Chapter 8, *Events*.

Related Commands

`QLength`, `XAllowEvents`, `XCheckIfEvent`, `XCheckMaskEvent`, `XCheckTyped-Event`, `XCheckTypedWindowEvent`, `XEventsQueued`, `XGetInputFocus`, `XGet-MotionEvents`, `XIfEvent`, `XMaskEvent`, `XNextEvent`, `XPeekEvent`, `XPeek-IfEvent`, `XPending`, `XPutBackEvent`, `XSelectInput`, `XSendEvent`, `XSetInput-Focus`, `XSynchronize`, `XWindowEvent`.

Name

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

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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.

For more information, see Volume One, Chapter 14, *Window Management*.

Errors

`BadValue`
`BadWindow`

Related Commands

`XCirculateSubwindowsDown`, `XCirculateSubwindowsUp`, `XConfigureWindow`, `XLowerWindow`, `XMoveResizeWindow`, `XMoveWindow`, `XQueryTree`, `XRaise-Window`, `XReparentWindow`, `XResizeWindow`, `XRestackWindows`.

XCirculateSubwindowsDown

Name

XCirculateSubwindowsDown — circulate the bottom child to the top of the stacking order.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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.

This function 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. This allows the window manager to intercept this request when *w* is the root window. Usually, only the window manager will call this on the root window.

For more information, see Volume One, Chapter 14, *Window Management*.

Errors

`BadWindow`

Related Commands

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

XCirculateSubwindowsUp

Name

XCirculateSubwindowsUp — circulate the top child to the bottom of the stacking order.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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. This allows the window manager to intercept this request when *w* is the root window. Usually, only the window manager will call this on the root window.

For more information, see Volume One, Chapter 14, *Window Management*.

Errors

`BadWindow`

Related Commands

`XCirculateSubwindows`, `XCirculateSubwindowsDown`, `XConfigureWindow`, `XLowerWindow`, `XMoveResizeWindow`, `XMoveWindow`, `XQueryTree`, `XRaise-Window`, `XReparentWindow`, `XResizeWindow`, `XRestackWindows`.

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 connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of an `InputOutput` window.
x *y*	Specify the x and y coordinates of the upper-left corner of the rectangle to be cleared, relative to the origin of the window.
width *height*	Specify the dimensions in pixels of the rectangle to be cleared.
exposures	Specifies whether exposure events are generated. Must be either `True` or `False`.

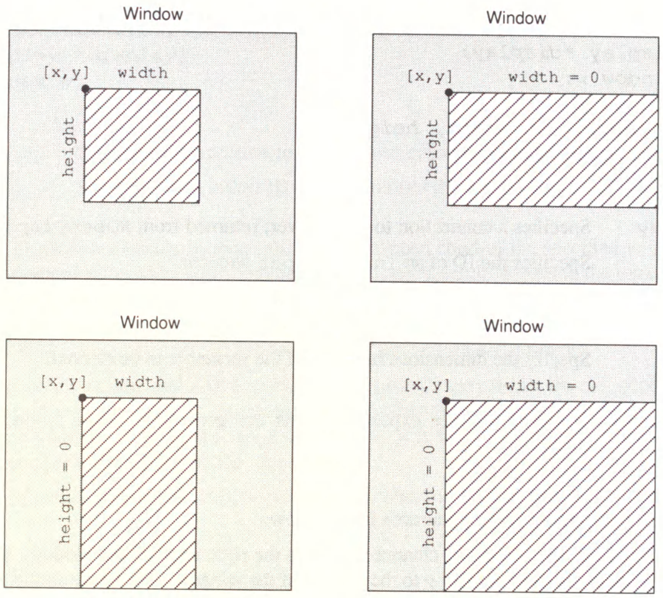
Description

`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. See figure above..

If the window has a defined background tile or it is `ParentRelative`, the rectangle is tiled with a `plane_mask` of all 1's, a `function` of `GXcopy`, and a `subwindow_mode` of `ClipByChildren`. 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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.



Errors

BadMatch	Window is an InputOnly class window.
BadValue	
BadWindow	

Related Commands

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

Name

XClearWindow — clear an entire window.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window to be cleared.

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 1's and `function` of `GXcopy`. If the window has background `None`, the contents of the window are not changed.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

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

Related Commands

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

XClipBox

Name

XClipBox — generate the smallest rectangle enclosing a region.

Synopsis

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

Arguments

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

Description

`XClipBox` returns the smallest rectangle that encloses the given region.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Structures

`Region` is a pointer to an opaque structure type.

Related Commands

`XCreateRegion`, `XDestroyRegion`, `XEmptyRegion`, `XEqualRegion`, `XIntersectRegion`, `XOffsetRegion`, `XPointInRegion`, `XPolygonRegion`, `XRectInRegion`, `XSetRegion`, `XShrinkRegion`, `XSubtractRegion`, `XUnion-RectWithRegion`, `XUnionRegion`, `XXorRegion`.

XCloseDisplay

Name

XCloseDisplay — disconnect a client program from an X server and display.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 close down mode 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 requests that have been buffered but not yet sent to the server.

Although these operations automatically (implicitly) occur when a process exits under UNIX, you should call `XCloseDisplay` anyway.

For more information, see Volume One, Chapter 3, *Basic Window Program.*

Related Commands

`DefaultScreen`, `XFree`, `XNoOp`, `XOpenDisplay`.

XConfigureWindow

Name

XConfigureWindow — change the 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 connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window to be reconfigured.
value_mask	Specifies which values are to be set using information in the `values` structure. *value_mask* is the bitwise OR of any number of symbols listed 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. If selected, a `ConfigureNotify` event is generated to announce any changes.

If the window to be reconfigured is a top-level window, there will be interaction with the window manager if the `override_redirect` attribute of the window is `False`. In this case, the X server sends a `ConfigureRequest` event to the window manager and does not reconfigure the window. The window manager receives this event and then makes the decision whether to allow the application to reconfigure its window. The client should wait for the `ConfigureNotify` event to find out the size and position of the window.

In Release 4, `XReconfigureWMWindow` should be used instead of `XConfigureWindow` for top-level windows. This routine handles restacking of top-level windows properly.

If a window's size actually changes, the window's subwindows may move according to their window gravity. If they do, `GravityNotify` events will be generated for them. Depending on the window's bit gravity, the contents of the window also may be moved. See Volume One, Chapter 4, *Window Attributes* 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 afterward 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 of the upper-left outer corner of the window relative to the parent's origin.
width *height*	Specify the inside size of the window in pixels, 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 window *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. If *w* occludes *sibling*, then *w* is placed at the bottom of the stack. If *w* and *sibling* do not overlap, no change is made.

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

Under Release 4, use `XReconfigureWMWindow` to configure a top-level window.

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`	Attempt to set any invalid attribute of `InputOnly` window. *`sibling`* specified without a *`stack_mode`*. The *`sibling`* window is not actually a sibling.
`BadValue`	*`width`* or *`height`* is 0.
`BadWindow`	

Related Commands

`XCirculateSubwindows`, `XCirculateSubwindowsDown`, `XCirculate-SubwindowsUp`, `XLowerWindow`, `XMoveResizeWindow`, `XMoveWindow`, `XQuery-Tree`, `XReconfigureWMWindow`, `XRaiseWindow`, `XReparentWindow`, `XResize-Window`, `XRestackWindows`.

XConvertSelection

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 connection to an X server; returned from `XOpenDisplay`.
selection	Specifies the selection atom. `XA_PRIMARY` and `XA_SECONDARY` are the standard selection atoms.
target	Specifies the atom of the type property that specifies the desired format for the data.
property	Specifies the property in which the requested data is to be placed. `None` is also valid, but current conventions specify that the requestor is in a better position to select a property than the selection owner.
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 property to place the information in (*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 or the owner sends a `SelectionNotify` event to the *requestor* with property `None`. Whether or not the owner exists, the arguments are passed unchanged. See Volume One, Chapter 10, *Interclient Communication*, for a description of selection events and selection conventions.

Errors

`BadAtom`
`BadWindow`

Related Commands

`XGetSelectionOwner`, `XSetSelectionOwner`.

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 connection to an X server; 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 upper-left corner of the source rectangle relative to the origin of the source drawable.
width *height*	Specify the dimensions in pixels 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 1's and function `GXcopy`) with that background. Regardless of tiling, if the destination is a window and `graphics_exposures` in *gc* is `True`, then `GraphicsExpose` events for all corresponding destination regions are generated. If `graphics_exposures` is `True` but no regions are exposed, then a `NoExpose` event is generated.

If regions of the source rectangle are not obscured and `graphics_exposures` 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

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

Related Commands

XClearArea, XClearWindow, XCopyPlane, XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoint, XDrawPoints, XDraw-Rectangle, XDrawRectangles, XDrawSegments, XFillArc, XFillArcs, XFill-Polygon, XFillRectangle, XFillRectangles.

XCopyColormapAndFree

Name

XCopyColormapAndFree — copy a colormap and return a new colormap ID.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the colormap you are moving out of.

Description

`XCopyColormapAndFree` is used to obtain a new virtual colormap when allocating colorcells 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*. 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 by the client 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 that have not been freed since they were allocated. Values in other entries of the new `Colormap` are undefined.

For more information, see Volume One, Chapter 7, *Color*.

Errors

`BadAlloc`
`BadColormap`

Related Commands

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

XCopyGC

Name

XCopyGC — copy a graphics context.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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. *valuemask* is made by combining any number of the mask symbols listed in the Structures section using bitwise OR (|).
dest	Specifies the destination graphics context.

Description

`XCopyGC` copies the selected elements of one graphics context to another. See Volume One, Chapter 5, *The Graphics Context*, 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;

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

Related Commands

DefaultGC, XChangeGC, XCreateGC, XFreeGC, XGContextFromGC, XGet-GCValues, XSetArcMode, XSetBackground, XSetClipMask, XSetClipOrigin, XSetClipRectangles, XSetDashes, XSetFillRule, XSetFillStyle, XSet-Foreground, XSetFunction, XSetGraphicsExposures, XSetLineAttributes, XSetPlaneMask, XSetState, XSetStipple, XSetSubwindowMode, XSet-TSOrigin.

XCopyPlane

Name

XCopyPlane — copy a single plane of a drawable into a drawable with depth, applying pixel values.

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 connection to an X server; 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 upper-left corner of the source rectangle relative to the origin of the drawable.
width *height*	Specify the width and height in pixels. These are the dimensions of both the source and destination rectangles.
dest_x *dest_y*	Specify the x and y coordinates at which the copied area will be placed relative to the origin of the destination drawable.
plane	Specifies the source bit-plane. You must set exactly one bit, and the bit must specify a plane that exists in *src*.

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`.

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

For more information, see Volume One, Chapter 5, *The Graphics Context*.

Errors

`BadDrawable`	
`BadGC`	
`BadMatch`	*`src`* and *`dest`* do not have the same root.
`BadValue`	*`plane`* does not have exactly one bit set, or bit specified in *`plane`* is not a plane in *`src`*.

Related Commands

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

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.

For more information on association tables, see Volume One, Appendix B, *X10 Compatibility*.

Structures

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

Related Commands

`XDeleteAssoc`, `XDestroyAssocTable`, `XLookUpAssoc`, `XMakeAssoc`.

XCreateBitmapFromData

Name

XCreateBitmapFromData — create a bitmap from `X11` bitmap format data.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies a drawable. This determines which screen to create the bitmap on.
data	Specifies the bitmap data, in X11 bitmap file format.
width *height*	Specify the dimensions in pixels of the created bitmap. If smaller than the bitmap data, the upper-left corner of the data 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 as shown below (it cannot be in X10 format). The following format is assumed for the data, where the variables are members of the `XImage` structure described in Volume One, Chapter 6, *Drawing Graphics and Text*:

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

`XCreateBitmapFromData` creates an image with the specified data and copies it into the created pixmap. 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[] = {
    0xf8, 0x1f, 0xe3, 0xc7, 0xcf, 0xf3, 0x9f, 0xf9,
    0xbf, 0xfd, 0x33, 0xcc, 0x7f, 0xfe, 0x7f, 0xfe,
          0x7e, 0x7e, 0x7f, 0xfe, 0x37, 0xec, 0xbb, 0xdd,
          0x9c, 0x39, 0xcf, 0xf3, 0xe3, 0xc7, 0xf8, 0x1f};

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

If the call could not create a pixmap of the requested size on the server, `XCreateBitmap-FromData` returns `0` (zero), and the server generates a `BadAlloc` error. If the requested depth is not supported on the screen of the specified drawable, the server generates a `Bad-Match` error.

The user should free the bitmap using `XFreePixmap` when it is no longer needed.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

`BadAlloc`	Server has insufficient memory to create bitmap.
`BadDrawable`	
`BadValue`	Specified bitmap dimensions are zero.

Related Commands

`XCreatePixmap`, `XCreatePixmapFromBitmapData`, `XCreatePixmapFrom-BitmapData`, `XFreePixmap`, `XQueryBestSize`, `XQueryBestStipple`, `XQuery-BestTile`, `XReadBitmapFile`, `XSetTile`, `XSetWindowBackgroundPixmap`, `XSetWindowBorderPixmap`, `XWriteBitmapFile`.

XCreateColormap

Name

XCreateColormap — create a colormap.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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`, `Alloc-ColorPlanes` and `XStoreColors`. If you pass the constant `AllocAll`, the entire colormap is allocated writable (all the entries are read/write, nonshareable and have undefined initial RGB 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 a call to the function `XAllocColor` cells 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 0 and mask values the same as the `red_mask`, `green_mask`, and `blue_mask` members in *visual*.

The *visual* argument should be as returned from the DefaultVisual macro, XMatch-VisualInfo, or XGetVisualInfo.

If the hardware colormap on the server is immutable, and therefore there is no possibility that a virtual colormap could ever be installed, XCreateColormap returns the default colormap. Code should check the returned ID against the default colormap to catch this situation.

For more information on creating colormaps, see Volume One, Chapter 7, *Color*.

Errors

BadAlloc	
BadMatch	Didn't use AllocNone for StaticColor, StaticGray, or True-Color. *visual* type not supported on screen.
BadValue	
BadWindow	

Related Commands

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

Name

XCreateFontCursor — create a cursor from the standard cursor font.

Synopsis

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

Arguments

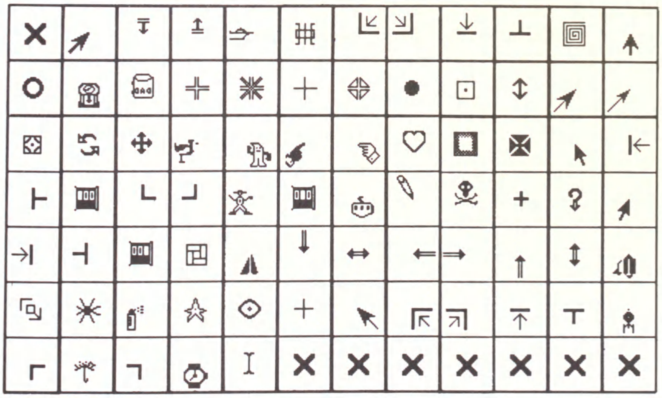
display	Specifies a connection to an X server; returned from `XOpenDisplay`.
shape	Specifies which character in the standard cursor font should be used for the cursor.

Description

X provides a set of standard cursor shapes in a special font named "cursor." Programs are encouraged to use this interface for their cursors, since the font can be customized for the individual display type and shared 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 more information about cursors and their shapes in fonts, see Appendix I, *The Cursor Font*.



Errors

`BadAlloc`	
`BadFont`	
`BadValue`	The *shape* argument does not specify a character in the standard cursor font.

Related Commands

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

XCreateGC

Name

XCreateGC — create a new graphics context for a given screen with the depth of the specified drawable.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies a drawable. The created GC can only be used to draw in drawables of the same depth as this *drawable*.
valuemask	Specifies which members of the GC are to be set using information in the *values* structure. *valuemask* is made by combining any number of the mask symbols listed in the Structures section.
values	Specifies a pointer to an `XGCValues` structure which will provide components for the new GC.

Description

`XCreateGC` creates a new graphics context resource in the server. The returned GC can be used in subsequent drawing requests, but only on drawables on the same screen and of the same depth as the drawable specified in the *drawable* argument.

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
`plane_mask`	all 1's
`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 filled with foreground pixel
`stipple`	Pixmap filled with 1's

Component	Value
`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])

An application should minimize the number of GCs it creates, because some servers cache a limited number of GCs in the display hardware, and can attain better performance with a small number of GCs.

For more information, see Volume One, Chapter 5, *The Graphics Context*.

Errors

`BadAlloc`	Server could not allocate memory for GC.
`BadDrawable`	Specified drawable is invalid.
`BadFont`	Font specified for *`font`* component of GC has not been loaded.
`BadMatch`	Pixmap specified for *`tile`* component has different depth or is on different screen from the specified drawable. Or pixmap specified for `stipple` or `clip_mask` component has depth other than 1.
`BadPixmap`	Pixmap specified for *`tile`*, *`stipple`*, or `clip_mask` components is invalid.
`BadValue`	Values specified for *`function`*, `line_style`, `cap_style`, `join_style`, `fill_style`, `fill_rule`, `subwindow_mode`, `graphics_exposures`, `dashes`, or `arc_mode` are invalid, or invalid mask specified for *`valuemask`* argument.

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;                  /* 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)
```

Related Commands

DefaultGC, XChangeGC, XCopyGC, XFreeGC, XGContextFromGC, XGetGCValues, XSetArcMode, XSetBackground, XSetClipMask, XSetClipOrigin, XSetClip-Rectangles, XSetDashes, XSetFillRule, XSetFillStyle, XSetForeground, XSetFunction, XSetGraphicsExposures, XSetLineAttributes, XSetPlane-Mask, XSetState, XSetStipple, XSetSubwindowMode, XSetTSOrigin.

XCreateGlyphCursor

Name

XCreateGlyphCursor — create a cursor from font glyphs.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
source_font	Specifies the font from which a character is to be used for the cursor.
mask_font	Specifies the mask font. Optional; specify 0 if not needed.
source_char	Specifies the index into the cursor shape font.
mask_char	Specifies the index into the mask shape font. Optional; specify 0 if not needed.
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 characters, perhaps in separate fonts. The mask font and character are optional. If *mask_char* is not specified, all pixels of the source are displayed.

The x offset for the hotspot of the created cursor is the left-bearing for the source character, and the y offset is the ascent, each measured from the upper-left corner of the bounding rectangle of the character.

The origins of the source and mask (if it is defined) characters 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, *source_char* and *mask_char* 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.

For more information on fonts and cursors, see Volume One, Chapter 6, *Drawing Graphics and Text*.

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).

Related Commands

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

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 connection to an X server; returned from `XOpenDisplay`.
visual	Specifies a pointer to a visual that should match the visual of the window the image is to be displayed in.
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 beginning of the data (pointed to by *data*) 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 *height*	Specify the width and height in pixels of the image.
bitmap_pad	Specifies the quantum of a scan line. In other words, the start of one scan line is separated in client memory from the start of the next scan line 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 scan line and the start of the next. If you pass a value of 0 here, Xlib assumes that the scan lines 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. It initializes the structure with byte order, bit order, and bitmap unit values, and returns a pointer to the XImage structure. The red, green, and blue mask values are defined for ZPixmap format images only and are derived from the Visual structure passed in.

For a description of images, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Related Commands

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

XCreatePixmap

Name

XCreatePixmap — create a pixmap.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable. May be an `InputOnly` window.
width *height*	Specify the width and height in pixels of the pixmap. The values must be nonzero.
depth	Specifies the depth of the pixmap. The depth must be supported by the screen of the specified drawable. (Use `XListDepths` if in doubt.)

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 drawn drawn into with GCs of the same depth, and can only be copied to 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.

Pixmaps should be considered a precious resource, since many servers have limits on the amount of off-screen memory available.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

`BadAlloc`	
`BadDrawable`	
`BadValue`	*width* or *height* is 0. *depth* is not supported on screen.

Related Commands

`XCreateBitmapFromData`, `XCreatePixmapFromBitmapData`, `XFreePixmap`, `XListDepths`, `XListPixmapFormat`, `XQueryBestCursor`, `XQueryBestSize`, `XQueryBestStipple`, `XQueryBestTile`, `XReadBitmapFile`, `XSetTile`, `XSetWindowBackgroundPixmap`, `XSetWindowBorderPixmap`, `XWriteBitmapFile`.

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 connection to an X server; returned from `XOpenDisplay`.
source	Specifies the shape of the source cursor. A pixmap of depth 1.
mask	Specifies the bits of the cursor that are to be displayed (the mask or stipple). A 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*	Specify the coordinates of the cursor's hotspot relative to the source's origin. 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 1 bits in the source, and the background is used for the 0 bits. Both source and mask (if specified) must have depth 1, but can have any root. The mask pixmap defines the shape of the cursor; that is, the 1 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 the source.

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

For more information on cursors, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Structures

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

```
        char pad;
    } XColor;
```

Errors

`BadAlloc`	
`BadMatch`	Mask bitmap must be the same size as source bitmap.
`BadPixmap`	

Related Commands

`XCreateBitmapFromData`, `XDefineCursor`, `XCreateFontCursor`, `XCreate-Pixmap`, `XCreatePixmapCursor`, `XFreeCursor`, `XFreePixmap`, `XQueryBest-Cursor`, `XQueryBestCursor`, `XQueryBestSize`, `XQueryBestSize`, `XRead-BitmapFile`, `XRecolorCursor`, `XUndefineCursor`.

XCreatePixmapFromBitmapData

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 connection to an `Display` structure, returned from `XOpen-Display`.
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*	Specify 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, where the variables are members of the `XImage` structure described in Volume One, Chapter 6, *Drawing Graphics and Text*:

```
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[] = {
    0xf8, 0x1f, 0xe3, 0xc7, 0xcf, 0xf3, 0x9f, 0xf9, 0xbf,
    0xfd, 0x33, 0xcc, 0x7f, 0xfe, 0x7f, 0xfe, 0x7e, 0x7e,
    0x7f, 0xfe, 0x37, 0xec, 0xbb, 0xdd, 0x9c, 0x39, 0xcf,
    0xf3, 0xe3, 0xc7, 0xf8, 0x1f};
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, using images.

Pixmaps should be considered a precious resource, since many servers have limits on the amount of off-screen memory available.

Errors

`BadAlloc`	
`BadDrawable`	
`BadValue`	The *`width`* or *`height`* of pixmap are zero, or *`depth`* is not a valid depth on the screen specified by `drawable`.

Related Commands

`XCreateBitmapFromData`, `XCreateFontCursor`, `XCreatePixmap`, `XCreate-PixmapCursor`, `XDefineCursor`, `XFreeCursor`, `XFreePixmap`, `XListPixmap-Formats`, `XQueryBestCursor`, `XQueryBestSize`, `XReadBitmapFile`, `XRecolor-Cursor`, `XUndefineCursor`.

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 a defined shape and size. Many of the functions that perform operations on regions can also create regions.

For a description of regions, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Structures

`Region` is a pointer to an opaque structure type.

Related Commands

`XClipBox`, `XDestroyRegion`, `XEmptyRegion`, `XEqualRegion`, `XIntersect-Region`, `XOffsetRegion`, `XPointInRegion`, `XPolygonRegion`, `XRectInRegion`, `XSetRegion`, `XShrinkRegion`, `XSubtractRegion`, `XUnionRectWithRegion`, `XUnionRegion`, `XXorRegion`.

Name

XCreateSimpleWindow — create 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 upper-left pixel of the new window's border relative to the origin of the parent (inside the parent window's border).
width *height*	Specify the width and height, in pixels, 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` if you want to set the window attributes while creating a window. (After creation, `XChangeWindowAttributes` can be used.)

`XCreateSimpleWindow` returns the 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 `MapWindow` to display it. This function generates a `XCreateNotify` 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` or `XChangeWindowAttributes`.

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

For more information, see Volume One, Chapter 2, *X Concepts*, and Volume One, Chapter 3, *Basic Window Program.*

Errors

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

Related Commands

`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 connection to an X server; 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 of the upper-left pixel of the new window's border relative to the origin of the parent (upper left inside the parent's border).
width *height*	Specify the width and height, in pixels, of the window. 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 the server generates a `BadValue` error.
border_width	Specifies the width, in pixels, of the new window's border. Must be 0 for `InputOnly` windows, otherwise a `BadMatch` error is generated.
depth	Specifies the depth of the window, which is less than or equal to the parent's depth. A depth of `CopyFromParent` means the depth is taken from the parent. Use `XListDepths` is choosing an unusual depth. The specified depth paired with the *visual* argument must be supported on the screen.
class	Specifies the new window's class. Pass one of these constants: `InputOutput`, `InputOnly`, or `CopyFromParent`.
visual	Specifies a connection to an visual structure describing the style of colormap to be used with this window. `CopyFromParent` is valid.
valuemask	Specifies which window attributes are defined in the *attributes* argument. If *valuemask* is 0, *attributes* is not referenced. This mask is the bitwise OR of the valid attribute mask bits listed in the Structures section below.

`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 use `XCreateWindow` or `XCreateSimpleWindow`. `XCreateWindow` is a more general function that allows you to set specific window attributes when you create the window. 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 only `InputOutput` windows that use the default depth and visual.

`XCreateWindow` returns the 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.

The visual should be `DefaultVisual` or one returned by `XGetVisualInfo` or `XMatchVisualInfo`. The depth should be `DefaultDepth`, 1, or a depth returned by `XListDepths`. In current implementations of Xlib, if you specify a visual other than the one used by the parent, you must first find (using `XGetRGBColormaps`) or create a colormap matching this visual and then set the colormap window attribute in the *`attributes`* and *`valuemask`* arguments. Otherwise, you will get a `BadMatch` error.

For more information, see Volume One, Chapter 4, *Window Attributes*.

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;
```

```
/* 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, do_not_propagate_mask, override_redirect or cursor specified for InputOnly window.
BadColormap	*depth* nonzero for InputOnly.
BadCursor	Parent of InputOutput is InputOnly.
BadMatch	*border_width* is nonzero for InputOnly.
BadPixmap	*depth* not supported on screen for InputOutput.
BadValue	*width* or *height* is 0.
BadWindow	*visual* not supported on screen.

Related Commands

XCreateSimpleWindow, XDestroySubwindows, XDestroyWindow, XListDepths.

XDefineCursor

Name

XDefineCursor — assign a cursor to a window.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window in which the cursor is to be displayed.
cursor	Specifies the cursor to be displayed when the pointer is in the specified window. Pass `None` to have the parent's cursor displayed in the window, or for the root window, to have the default cursor 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.

For more information on available cursors, see Appendix I, *The Cursor Font*.

Errors

`BadCursor`
`BadWindow`

Related Commands

`XCreateFontCursor`, `XCreateGlyphCursor`, `XCreatePixmapCursor`, `XFree-Cursor`, `XQueryBestCursor`, `XQueryBestSize`, `XRecolorCursor`, `XUndefine-Cursor`.

XDeleteAssoc

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 connection to an X server; returned from `XOpenDisplay`.
table	Specifies one of the association tables created by `XCreateAssocTable`.
x_id	Specifies the X resource ID of the association to be deleted.

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 nonexistent `XID`'s) are meaningless and cause no problems. Deleting associations in no way impairs the performance of an `XAssocTable`.

For more information on association tables, see Volume One, Appendix B, *X10 Compatibility*.

Structures

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

Related Commands

`XCreateAssocTable`, `XDestroyAssocTable`, `XLookUpAssoc`, `XMakeAssoc`.

XDeleteContext

Name

XDeleteContext — delete a context entry for a given window and type.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 function returns `XCNOENT` if the context could not be found, or zero if it succeeds. `XDeleteContext` does not free the memory allocated for the data whose address was saved.

See Volume One, Chapter 13, *Other Programming Techniques*, for a description of context management.

Structures

```
typedef int XContext;
```

Related Commands

`XFindContext`, `XSaveContext`, `XUniqueContext`.

XDeleteModifiermapEntry

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` itself does not change the mapping.

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 after this call.

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
```

Related Commands

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

XDeleteProperty

Name

XDeleteProperty — delete a window property.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window whose property you want to delete.
property	Specifies the atom of the property to be deleted.

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 to set the property once again. If the property was defined on the specified window, `XDeleteProperty` generates a `PropertyNotify` event.

See the introduction to properties in Volume One, Chapter 2, *X Concepts*, or more detailed information in Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadAtom`
`BadWindow`

Related Commands

`XChangeProperty`, `XGetAtomName`, `XGetFontProperty`, `XGetWindowProperty`, `XInternAtom`, `XListProperties`, `XRotateWindowProperties`, `XSetStandard-Properties`.

XDestroyAssocTable

Name

XDestroyAssocTable — free the memory allocated for an association table.

Synopsis

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

Arguments

table	Specifies the association table whose memory is to be freed.

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 consequences.

For more information on association tables, see Volume One, Appendix B, *X10 Compatibility*.

Structures

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

Related Commands

`XCreateAssocTable`, `XDeleteAssoc`, `XLookUpAssoc`, `XMakeAssoc`.

XDestroyImage

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. (If the image data is statically allocated, the pointer to the data in the `XImage` structure must be set to zero before calling `XDestroyImage`.)

For more information on images, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Related Commands

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

XDestroyRegion

Name

XDestroyRegion — deallocate storage associated with a region.

Synopsis

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

Arguments

r	Specifies the region to be destroyed.

Description

`XDestroyRegion` frees the memory associated with a region and invalidates pointer *r*.

See Volume One, Chapter 6, *Drawing Graphics and Text*, for a description of regions.

Related Commands

`XClipBox`, `XCreateRegion`, `XEmptyRegion`, `XEqualRegion`, `XIntersect-Region`, `XOffsetRegion`, `XPointInRegion`, `XPolygonRegion`, `XRectInRegion`, `XSetRegion`, `XShrinkRegion`, `XSubtractRegion`, `XUnionRectWithRegion`, `XUnionRegion`, `XXorRegion`.

XDestroySubwindows

Name

XDestroySubwindows — destroy all subwindows of a window.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window whose subwindows are to be destroyed.

Description

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

`XDestroySubwindows` generates exposure events on window *w*, if any mapped subwindows were actually destroyed. This is much more efficient than deleting many subwindows one at a time, since 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).

Never call `XDestroySubwindows` with the window argument set to the root window! This will destroy all the applications on the screen, and if there is only one screen, often the server as well.

Errors

`BadWindow`

Related Commands

`XCreateSimpleWindow`, `XCreateWindow`, `XDestroyWindow`.

XDestroyWindow

Name

XDestroyWindow — unmap and destroy a window and all subwindows.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
window	Specifies the ID of the window to be destroyed.

Description

If *window* is mapped, an `UnmapWindow` request is performed automatically. The window and all inferiors (recursively) 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. `XDestroyWindow` may also generate `EnterNotify` events if *window* was mapped and contained the pointer.

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

Errors

`BadWindow`

Related Commands

`XCreateSimpleWindow`, `XCreateWindow`, `XDestroySubwindows`.

XDisableAccessControl

Name

XDisableAccessControl — allow access from any host.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.

Description

`XDisableAccessControl` instructs the server to allow access from clients on any host. This disables use of the host access list.

This routine can only be called from a client running on the same host as the server.

For more information on access control, see Volume One, Chapter 13, *Other Programming Techniques*.

Errors

`BadAccess`

Related Commands

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

XDisplayKeycodes

Name

XDisplayKeycodes — obtain the range of legal keycodes for a server.

Synopsis

```
XDisplayKeycodes(display, min_keycodes, max_keycodes)
        Display *display;
        int *min_keycode, *max_keycode;   /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
min_keycode	Returns the minimum keycode.
max_keycode	Returns the maximum keycode.

Description

`XDisplayKeycodes` returns the *min_keycode* and *max_keycode* supported by the specified server. The minimum keycode returned is never less than 8, and the maximum keycode returned is never greater than 255. Not all keycodes in this range are required to have corresponding keys.

For more information, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Related Commands

`XKeycodeToKeysym`, `XKeysymToKeycode`, `XLookupString`.

XDisplayName

Name

XDisplayName — report the display name (when connection to a 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 `XOpenDisplay`. 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 DISPLAY environment variable 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 server the program attempted to connect to.

For more information, see Volume One, Chapter 3, *Basic Window Program*.

Related Commands

`XGetErrorDatabaseText`, `XGetErrorText`, `XSetAfterFunction`, `XSetError-Handler`, `XSetIOErrorHandler`, `XSynchronize`.

Name

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

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
vlist	Specifies a pointer to the list of vertices that indicates 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*. Its performance is likely to be low.

`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 achieves the effects of the X10 XDraw, XDrawDashed, and XDrawPatterned functions.

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_x_origin, ts_y_origin, dash_offset, and dash_list.

A Status of zero is returned on failure, and nonzero on success.

For more information, see Volume One, Appendix B, *X10 Compatibility*.

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 */
```

Related Commands

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

Name

XDrawArc — draw an arc fitting inside a rectangle.

Synopsis

```
XDrawArc(display, drawable, gc, x, y, width, height,
         angle1, angle2)
    Display *display;
    Drawable drawable;
    GC gc;
    int x, y;
    unsigned int width, height;
    int angle1, angle2;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the upper-left corner of the rectangle that contains the arc, relative to the origin of the specified drawable.
width *height*	Specify the width and height in pixels of 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 64ths of a degree (360 * 64 is a complete circle).
angle2	Specifies the end of the arc relative to the start of the arc. Angles are specified in 64ths of a degree (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, and the major and minor axes are specified by the `width` and `height`, respectively. The angles are signed integers in 64ths of a degree, with positive values indicating counterclockwise motion and negative values 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, and the path and extent of the arc is specified by *angle2* relative to the start of the arc.

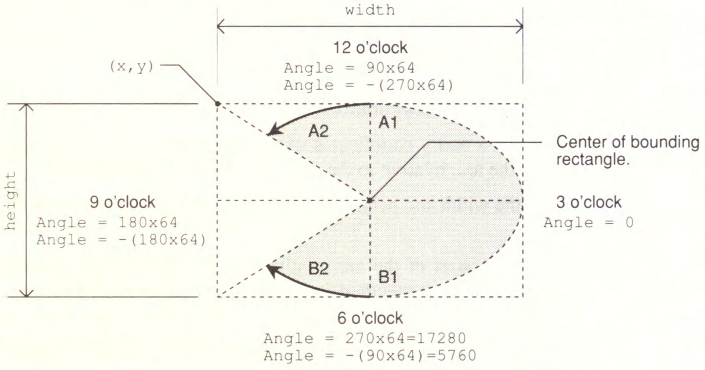
By specifying one axis to be zero, a horizontal or vertical line is drawn (inefficiently).

Angles are computed based solely on the coordinate system and ignore the aspect ratio. In other words, if the bounding rectangle of the arc is not square and *angle1* is zero and *angle2* is (`45x64`), a point drawn from the center of the bounding box through the endpoint of the arc will not pass through the corner of the rectangle.

For any given arc, no pixel is drawn more than once, even if `angle2` is greater than `angle1` by more than 360 degrees.

`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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.



Example 1:
Arc from A1 to A2, Counterclockwise
A1 = 90 X 64
A2 = 45 X 64

Example 2:
Arc from B1 to B2, Clockwise
B1 = 270 X 64
B2 = -(45 X 64)

Errors

`BadDrawable`
`BadGC`
`BadMatch`

Related Commands

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

XDrawArcs

Name

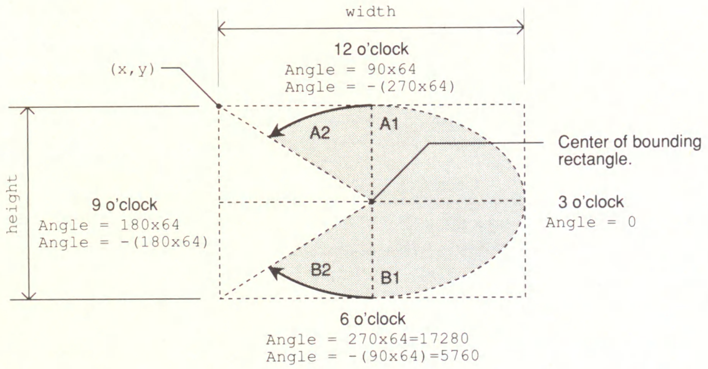
XDrawArcs — draw multiple arcs.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	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.



Example 1:
Arc from A1 to A2, Counterclockwise
A1 = 90 X 64
A2 = 45 X 64

Example 2:
Arc from B1 to B2, Clockwise
B1 = 270 X 64
B2 = -(45 X 64)

Description

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

There is a limit to the number of arcs that can be drawn in a single call. It varies according to the server. To determine how many arcs you can draw in a single call, find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and divide by three: this is the maximum number of arcs you can draw in a single `XDrawArcs` call.

The arcs are drawn in the order listed in the *`arcs`* array.

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

For any given arc, no pixel is drawn more than once. 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. If two arcs join correctly and if `line_width` is greater than 0 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.

`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`.

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 64ths of a degree) 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]
```

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Structures

```
typedef struct {
     short x, y;
     unsigned short width, height;
     short angle1, angle2;            /*  Start and end of arc, in */
                                      /*  64ths of degrees */
} XArc;
```

Errors

```
BadDrawable
BadGC
BadMatch
```

Related Commands

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

XDrawFilled

Name

XDrawFilled — draw a filled polygon or curve from vertex list (from `X10`).

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	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 X Version 10 `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_x_origin`, `ts_y_origin`, `dash_offset`, `dash_list`, `fill_style` and `fill_rule`.

`XDrawFilled` returns a `Status` of zero on failure, and nonzero on success.

For more information, see Volume One, Appendix B, *X10 Compatibility*.

Related Commands

`XClearArea`, `XClearWindow`, `XCopyArea`, `XCopyPlane`, `XDraw`, `XDrawArc`, `XDrawArcs`, `XDrawLine`, `XDrawLines`, `XDrawPoint`, `XDrawPoints`, `XDraw-Rectangle`, `XDrawRectangles`, `XDrawSegments`, `XFillArc`, `XFillArcs`, `XFill-Polygon`, `XFillRectangle`, `XFillRectangles`.

XDrawImageString

Name

XDrawImageString — draw 8-bit image text characters.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the baseline starting position for the image text character, 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 draws both the foreground and the background of the characters. It draws the characters in the foreground and fills the bounding box with the background.

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

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Errors

```
BadDrawable
BadGC
BadMatch
```

Related Commands

`XDrawImageString16`, `XDrawString`, `XDrawString16`, `XDrawText`, `XDraw-Text16`, `XQueryTextExtents`, `XQueryTextExtents16`, `XTextExtents`, `XText-Extents16`, `XTextWidth`, `XTextWidth16`.

XDrawImageString16

Name

XDrawImageString16 — draw 16-bit image text characters.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the baseline starting position for the image text character, 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 draws both the foreground and the background of the characters. It draws the characters in the foreground and fills the bounding box with the background.

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

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

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

Errors

BadDrawable
BadGC
BadMatch

Related Commands

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

Name

XDrawLine — draw a line between two points.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x1 *y1* *x2* *y2*	Specify the coordinates of the endpoints 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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Errors

`BadDrawable`	Specified drawable is invalid.
`BadGC`	Specified GC is invalid, or does not match the depth of drawable.
`BadMatch`	Specified `drawable` is an `InputOnly` window.

Related Commands

`XClearArea`, `XClearWindow`, `XCopyArea`, `XCopyPlane`, `XDraw`, `XDrawArc`, `XDrawArcs`, `XDrawFilled`, `XDrawLines`, `XDrawPoint`, `XDrawPoints`, `XDraw-Rectangle`, `XDrawRectangles`, `XDrawSegments`, `XFillArc`, `XFillArcs`, `XFill-Polygon`, `XFillRectangle`, `XFillRectangles`.

Name

XDrawLines — draw multiple connected lines.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
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` draws a series of lines joined end-to-end.

It 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 multiline request were a single filled shape.

There is a limit to the number of lines that can be drawn in a single call, that varies according to the server. To determine how many lines you can draw in a single call, you find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and divide by two, and this is the maximum number of lines you can draw in a single `XDrawLines` call.

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` uses the following components of the specified graphics context to draw multiple connected lines in the specified drawable: `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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

```
typedef struct {
    short x, y;
} XPoint;
```

Errors

BadDrawable	Specified drawable is invalid.
BadGC	Specified GC is invalid, or does not match the depth of drawable.
BadMatch	Specified drawable is an InputOnly window.
BadValue	Invalid coordinate_mode.

Related Commands

XClearArea, XClearWindow, XCopyArea, XCopyPlane, XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawPoint, XDrawPoints, XDraw-Rectangle, XDrawRectangles, XDrawSegments, XFillArc, XFillArcs, XFill-Polygon, XFillRectangle, XFillRectangles.

XDrawPoint

Name

XDrawPoint — draw a point.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the point, relative to the origin of the drawable.

Description

`XDrawPoint` draws 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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Errors

`BadDrawable`
`BadGC`
`BadMatch`

Related Commands

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

XDrawPoints

Name

XDrawPoints — draw multiple points.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
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` draws one or more points into the specified drawable.

There is a limit to the number of points that can be drawn in a single call, that varies according to the server. To determine how many points you can draw in a single call, you find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and this is the maximum number of points you can draw in a single `XDrawPoints` call.

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

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

```
typedef struct {
    short x, y;
} XPoint;
```

Errors

BadDrawable
BadGC
BadMatch
BadValue

Related Commands

XClearArea, XClearWindow, XCopyArea, XCopyPlane, XDraw, XDrawArc, XDrawArcs, XDrawFilled, XDrawLine, XDrawLines, XDrawPoints, XDraw-Rectangle, XDrawRectangles, XDrawSegments, XFillArc, XFillArcs, XFill-Polygon, XFillRectangle, XFillRectangles.

Name

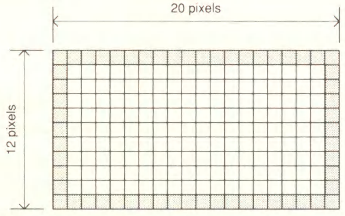
XDrawRectangle — draw an outline of a rectangle.

Synopsis

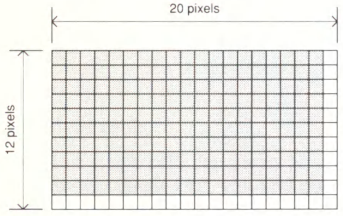
```
XDrawRectangle(display, drawable, gc, x, y, width, height)
    Display *display;
    Drawable drawable;
    GC gc;
    int x, y;
    unsigned int width, height;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the upper-left corner of the rectangle, relative to the drawable's origin.
width *height*	Specify the width and height in pixels. These dimensions define the outline of the rectangle.



`XDrawRectangle (display, drawable, gc, 0, 0, 19, 11);`



`XFillRectangle (display, drawable, gc, 0, 0, 19, 11);`

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_y_origin`, `dash_offset`, and `dash_list`.

For the specified rectangle, no pixel is drawn more than once.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structure

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

Errors

```
BadDrawable
BadGC
BadMatch
```

Related Commands

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

XDrawRectangles

Name

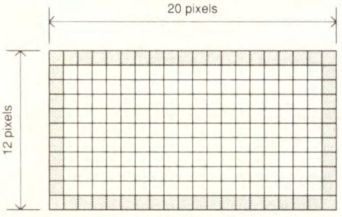
XDrawRectangles — draw the outlines of multiple rectangles.

Synopsis

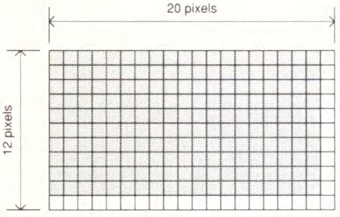
```
XDrawRectangles(display, drawable, gc, rectangles, nrectangles)
      Display *display;
      Drawable drawable;
      GC gc;
      XRectangle rectangles[];
      int nrectangles;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
rectangles	Specifies a pointer to an array of rectangles containing position and size information.
nrectangles	Specifies the number of rectangles in the array.



`XDrawRectangle (display, drawable, gc, 0, 0, 19, 11);`



`XFillRectangle (display, drawable, gc, 0, 0, 19, 11);`

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.

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.

There is a limit to the number of rectangles that can be drawn in a single call. It varies according to the server. To determine how many rectangles you can draw in a single call, find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and divide by two. This is the maximum number of rectangles you can draw in a single `XDrawRectangles` call.

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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

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

Errors

`BadDrawable`
`BadGC`
`BadMatch`

Related Commands

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

Name

XDrawSegments — draw multiple disjoint lines.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
segments	Specifies a pointer to an array of line 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`.

There is a limit to the number of segments that can be drawn in a single call. It varies according to the server. To determine how many segments you can draw in a single call, find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and divide by two. This is the maximum number of segments you can draw in a single `XDrawSegments` call.

`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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

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

Errors

`BadDrawable`	Specified drawable is invalid.
`BadGC`	Specified GC is invalid, or does not match the depth of drawable.
`BadMatch`	Specified *`drawable`* is an `InputOnly` window.

Related Commands

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

XDrawString

Name

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

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the baseline starting position for the character, relative to the origin of the specified drawable.
string	Specifies the character string.
length	Specifies the number of characters in *string*.

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 from which `lbearing`, `rbearing`, and `width` are measured.

`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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Errors

```
BadDrawable
BadFont
BadGC
BadMatch
```

Related Commands

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

XDrawString16

Name

XDrawString16 — draw two-byte text strings.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the baseline starting position for the character, 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 *string*.

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 from which `lbearing`, `rbearing`, and `width` are measured. For more information on text placement, see Volume One, Chapter 6, *Drawing Graphics and Text*.

`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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

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

Errors

`BadDrawable`
`BadFont`
`BadGC`
`BadMatch`

Related Commands

`XDrawImageString`, `XDrawImageString16`, `XDrawString`, `XDrawText`, `XDraw-Text16`, `XQueryTextExtents`, `XQueryTextExtents16`, `XTextExtents`, `XText-Extents16`, `XTextWidth`, `XTextWidth16`.

Name

XDrawText — draw 8-bit polytext strings.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of 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 *items* array.

Description

`XDrawText` is capable of drawing multiple strings on the same horizontal line 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 to fill the bounding box).

There is a limit to the number and size of strings that can be drawn in a single call, that varies according to the server. To determine how much text you can draw in a single call, you find out your server's maximum request size using `XMaxRequestSize`. Subtract four, and then subtract `((strlen(string) + 2) / 4)` for each string. This is the maximum amount of text you can draw in a single `XDrawText` call.

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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

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 don't change */
} XTextItem;
```

Errors

BadDrawable
BadFont
BadGC
BadMatch

Related Commands

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

XDrawText16

Name

XDrawText16 — draw 16-bit polytext strings.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of 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 two-byte characters.
nitems	Specifies the number of text items in the array.

Description

`XDrawText16` is capable of drawing multiple strings on the same horizontal line 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 drawing 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 to fill the bounding box).

There is a limit to the number and size of strings that can be drawn in a single call, that varies according to the server. To determine how much text you can draw in a single call, you find out your server's maximum request size using `XMaxRequestSize`. Subtract four, and then subtract `((strlen(string) + 2) / 4)` for each string. This is the maximum amount of text you can draw in a single `XDrawText16` call.

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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

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 don't change */
} XTextItem16;

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

Errors

BadDrawable
BadFont
BadGC
BadMatch

Related Commands

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

XEmptyRegion

Name

XEmptyRegion — determine if a region is empty.

Synopsis

```
Bool XEmptyRegion(r)
    Region r;
```

Arguments

r	Specifies the region to be checked.

Description

`XEmptyRegion` will return `True` if the specified region is empty, or `False` otherwise.

Structures

`Region` is a pointer to an opaque structure type.

Related Commands

`XClipBox`, `XCreateRegion`, `XDestroyRegion`, `XEqualRegion`, `XIntersect-Region`, `XOffsetRegion`, `XPointInRegion`, `XPolygonRegion`, `XRectInRegion`, `XSetRegion`, `XShrinkRegion`, `XSubtractRegion`, `XUnionRectWithRegion`, `XUnionRegion`, `XXorRegion`.

XEnableAccessControl

Name

XEnableAccessControl — use access control list to allow or deny connection requests.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.

Description

`XEnableAccessControl` instructs the server to use the host access list to determine whether access should be granted to clients seeking a connection with the server.

By default, the host access list is used. If access has not been disabled with `XDisableAccessControl` or `XSetAccessControl`, this routine does nothing.

This routine can only be called by clients running on the same host as the server.

For more information, see Volume One, Chapter 13, *Other Programming Techniques*.

Errors

`BadAccess`

Related Commands

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

XEqualRegion

Name

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

Synopsis

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

Arguments

r1	Specify the two regions you want to compare.
r2	

Description

`XEqualRegion` returns `True` if the two regions are identical; i.e., they have the same offset, size and shape, or `False` otherwise.

Regions are located using an offset from a 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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Structures

`Region` is a pointer to an opaque structure type.

Related Commands

`XClipBox`, `XCreateRegion`, `XDestroyRegion`, `XEmptyRegion`, `XIntersect-Region`, `XOffsetRegion`, `XPointInRegion`, `XPolygonRegion`, `XRectInRegion`, `XSetRegion`, `XShrinkRegion`, `XSubtractRegion`, `XUnionRectWithRegion`, `XUnionRegion`, `XXorRegion`.

Name

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

Synopsis

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

Arguments

display	Specifies a connection to a `Display` structure, returned from `XOpenDisplay`.
mode	Specifies whether the request buffer is flushed if there are no events in Xlib's queue. You can specify one of these constants: `QueuedAlready`, `QueuedAfterFlush`, `QueuedAfterReading`.

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 zero (it does not flush the request 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 request 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 request 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 (see Appendix C, *Macros*).

For more information, see Volume One, Chapter 8, *Events*.

Related Commands

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

Name

XFetchBuffer — return data from a cut buffer.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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. Specify an integer from 0 to 7 inclusive.

Description

`XFetchBuffer` returns data from one of the 8 buffers provided for interclient 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 by calling `XFree`. 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 preferred over cut buffers as a communication scheme.

For more information on cut buffers, see Volume One, Chapter 13, *Other Programming Techniques*.

Errors

`BadValue`	*buffer* not an integer between 0 and 7 inclusive.

Related Commands

`XFetchBytes`, `XRotateBuffers`, `XStoreBuffer`, `XStoreBytes`.

XFetchBytes

Name

XFetchBytes — return data from cut buffer 0.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 interclient 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 by calling `XFree`. 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 preferred over cut buffers as a communication method.

For more information on cut buffers, see Volume One, Chapter 13, *Other Programming Techniques*.

Related Commands

`XFetchBuffer`, `XRotateBuffers`, `XStoreBuffer`, `XStoreBytes`.

XFetchName

Name

XFetchName — get a window's name (XA_WM_NAME property).

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from XOpenDisplay.
w	Specifies the ID of 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 XA_WM_NAME property has not been set for this window, XFetchName sets *windowname* to NULL. When finished with it, a client can free the name string using XFree.

Description

XFetchName is superseded by XGetWMName in Release 4. XFetchName returns the current value of the XA_WM_NAME property for the specified window. XFetchName returns nonzero if it succeeds, and zero if the property has not been set for the argument window.

For more information, see Volume One, Chapter 10, *Interclient Communication*, and Chapter 14, *Window Management*.

Errors

BadWindow

Related Commands

XGetClassHint, XGetIconName, XGetIconSizes, XGetNormalHints, XGet-SizeHints, XGetTransientForHint, XGetWMHints, XGetZoomHints, XSet-ClassHint, XSetCommand, XSetIconName, XSetIconSizes, XSetNormalHints, XSetSizeHints, XSetTransientForHint, XSetWMHints, XSetZoomHints, XStoreName.

XFillArc

Name

XFillArc — fill an arc.

Synopsis

```
XFillArc(display, drawable, gc,  x, y, width, height,
        angle1, angle2)
    Display *display;
    Drawable drawable;
    GC gc;
    int x, y;
    unsigned int width, height;
    int angle1, angle2;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the upper-left corner of the bounding box containing the arc, relative to the origin of the drawable.
width *height*	Specify the width and height in pixels. 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 64ths of degrees.
angle2	Specifies the path and extent of the arc relative to the start of the arc. Angles are specified in 64ths of degrees.

Description

`XFillArc` draws a filled arc. 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 by `XFillArc` with the same arguments. See `XFillRectangle` for an example of the differences in pixels drawn by the draw and fill routines.

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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Errors

`BadDrawable`
`BadGC`
`BadMatch`

Related Commands

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

Name

XFillArcs — fill multiple arcs.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
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 filled arcs intersect, pixels will be drawn multiple times.

There is a limit to the number of arcs that can be filled in a single call, that varies according to the server. To determine how many arcs you can fill in a single call, you find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and divide by three, and this is the maximum number of arcs you can fill in a single `XFillArcs` call.

`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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

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

```
        short angle1, angle2;           /*  64ths of Degrees */
    } XArc;
```

Errors

BadDrawable
BadGC
BadMatch

Related Commands

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

XFillPolygon

Name

XFillPolygon — fill a polygon.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
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 `Coord-ModePrevious`.

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. See Volume One, Chapter 5, *The Graphics Context*, 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` means that for every pair of points inside the polygon, the line segment connecting them does not intersect the path. This can improve performance even more, but if the path is not convex, the graphics results are undefined.

Contiguous coincident points in the path are not treated as self-intersection.

`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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

```
typedef struct {
    short x, y;
} XPoint;
```

Errors

`BadDrawable`
`BadGC`
`BadMatch`
`BadValue`

Related Commands

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

XFillRectangle

Name

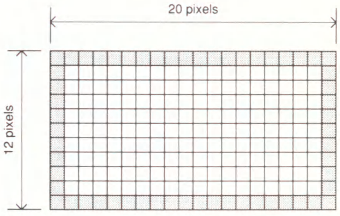
XFillRectangle — fill a rectangular area.

Synopsis

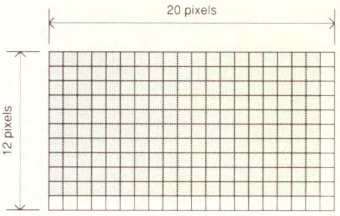
```
XFillRectangle(display, drawable, gc, x, y, width, height)
    Display *display;
    Drawable drawable;
    GC gc;
    int x, y;
    unsigned int width, height;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
x *y*	Specify the x and y coordinates of the upper-left corner of the rectangle, relative to the origin of the drawable.
width *height*	Specify the dimensions in pixels of the rectangle to be filled.



`XDrawRectangle (display, drawable, gc, 0, 0, 19, 11);`



`XFillRectangle (display, drawable, gc, 0, 0, 19, 11);`

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`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Errors

`BadDrawable`
`BadGC`
`BadMatch`

Related Commands

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

Name

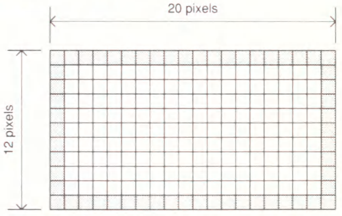
XFillRectangles — fill multiple rectangular areas.

Synopsis

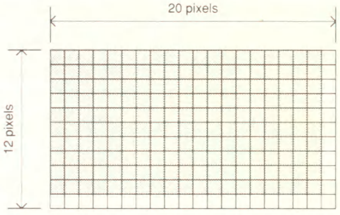
```
XFillRectangles(display, drawable, gc, rectangles, nrectangles)
      Display *display;
      Drawable drawable;
      GC gc;
      XRectangle *rectangles;
      int nrectangles;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable.
gc	Specifies the graphics context.
rectangles	Specifies a pointer to an array of rectangles.
nrectangles	Specifies the number of rectangles in the array.



`XDrawRectangle (display, drawable, gc, 0, 0, 19, 11);`



`XFillRectangle (display, drawable, gc, 0, 0, 19, 11);`

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.

There is a limit to the number of rectangles that can be filled in a single call, that varies according to the server. To determine how many rectangles you can fill in a single call, you find out your server's maximum request size using `XMaxRequestSize`. Subtract 3 and divide by two, and this is the maximum number of rectangles you can fill in a single `XDraw-Rectangles` call.

`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.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*, and Chapter 5, *The Graphics Context*.

Structures

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

Errors

BadDrawable
BadGC
BadMatch

Related Commands

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

XFindContext

Name

XFindContext — get data from the 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 connection to an X server; 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.

For more information on the context manager, see Volume One, Chapter 13, *Other Programming Techniques*.

Structures

```
typedef int XContext;
```

Related Commands

`XDeleteContext`, `XSaveContext`, `XUniqueContext`.

XFlush

Name

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

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.

Description

`XFlush` sends to the server ("flushes") all 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`, etc.), 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.

For more information, see Volume One, Chapter 2, *X Concepts*, and Chapter 3, *Basic Window Program*.

Related Commands

`XSync`.

XForceScreenSaver

Name

XForceScreenSaver — turn the screen saver on or off.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
mode	Specifies whether the 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 by calling `XSetScreenSaver` 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.

For more information on the screen saver, see Volume One, Chapter 13, *Other Programming Techniques*.

Errors

`BadValue`

Related Commands

`XActivateScreenSaver`, `XGetScreenSaver`, `XResetScreenSaver`, `XSetScreenSaver`.

XFree

Name

XFree — free specified memory allocated by an Xlib function.

Synopsis

```
XFree(data)
    caddr_t data;
```

Arguments

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

Description

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

Related Commands

`DefaultScreen`, `XCloseDisplay`, `XNoOp`, `XOpenDisplay`.

XFreeColormap

Name

XFreeColormap — delete a colormap and install the default colormap.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the colormap to delete.

Description

`XFreeColormap` destroys the specified 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 including the colormap ID.

`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 attribute for 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, since the default colormap is normally used.

For more information, see Volume One, Chapter 7, *Color*.

Errors

`BadColormap`

Related Commands

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

XFreeColors

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 connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the colormap.
pixels	Specifies an array of pixel values.
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 to have allocated those shared cells.

For more information, see Volume One, Chapter 7, *Color*.

Errors

`BadAccess`	Attempt to free a colorcell not allocated by this client (either unallocated or allocated by another client).
`BadColormap`	
`BadValue`	A pixel value is not a valid index into *cmap*.

Note: if more than one pixel value is in error, the one reported is arbitrary.

Related Commands

`BlackPixel`, `WhitePixel`, `XAllocColor`, `XAllocColorCells`, `XAllocColor-Planes`, `XAllocNamedColor`, `XLookupColor`, `XParseColor`, `XQueryColor`, `XQueryColors`, `XStoreColor`, `XStoreColors`, `XStoreNamedColor`.

Name

XFreeCursor — release a cursor.

Synopsis

```
XFreeCursor(display, cursor)
    Display *display;
    Cursor cursor;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
cursor	Specifies the ID of the cursor to be affected.

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 have this attribute set to `None` (which implies `CopyFromParent`). The specified cursor ID should not be referred to again.

Errors

`BadCursor`

Related Commands

`XCreateFontCursor`, `XCreateGlyphCursor`, `XCreatePixmapCursor`, `XDefineCursor`, `XQueryBestCursor`, `XQueryBestSize`, `XRecolorCursor`, `XUndefineCursor`.

XFreeExtensionList

Name

XFreeExtensionList — free memory allocated for a list of installed extensions.

Synopsis

```
XFreeExtensionList(list)
    char **list;
```

Arguments

list	Specifies a pointer to the list of extensions returned from `XListExtensions`.

Description

`XFreeExtensionList` frees the memory allocated by `XListExtensions`.

For more information, see Volume One, Chapter 13, *Other Programming Techniques*.

Related Commands

`XListExtensions`, `XQueryExtension`.

XFreeFont

Name

XFreeFont — unload a font and free storage for the font structure.

Synopsis

```
XFreeFont(display, font_struct)
    Display *display;
    XFontStruct *font_struct;
```

Arguments

display	Specifies a connection to an X server; 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 server unloads the font itself if no other client has loaded it.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

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 nonzero size*/
    unsigned default_char;         /* char to print for undefined character */
    int n_properties;              /* how many properties there are */
    XFontProp *properties;         /* pointer to array of additional properties*/
    XCharStruct min_bounds;        /* minimum bounds over all existing char*/
    XCharStruct max_bounds;        /* 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`

Related Commands

`XCreateFontCursor`, `XFreeFontInfo`, `XFreeFontNames`, `XFreeFontPath`, `XGetFontPath`, `XGetFontProperty`, `XListFonts`, `XListFontsWithInfo`, `XLoadFont`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XFreeFontInfo

Name

XFreeFontInfo — free the memory allocated by `XListFontsWithInfo`.

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 `XListFontsWithInfo`.
actual_count	Specifies the number of matched font names returned by `XListFontsWithInfo`.

Description

`XFreeFontInfo` frees the list of font information structures allocated by `XListFontsWithInfo`. It does not unload the specified fonts themselves.

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 nonzero size*/
     unsigned default_char;         /* char to print for undefined character */
     int n_properties;              /* how many properties there are */
     XFontProp *properties;         /* pointer to array of additional properties*/
     XCharStruct min_bounds;        /* minimum bounds over all existing char*/
     XCharStruct max_bounds;        /* 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;
```

Related Commands

`XCreateFontCursor`, `XFreeFont`, `XFreeFontNames`, `XGetFontPath`, `XGetFontProperty`, `XListFonts`, `XListFontsWithInfo`, `XLoadFont`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XFreeFontNames

Name

XFreeFontNames — free the memory allocated by `XListFonts`.

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`.

Related Commands

`XCreateFontCursor`, `XFreeFont`, `XFreeFontInfo`, `XFreeFontPath`, `XGetFont-Path`, `XGetFontProperty`, `XListFonts`, `XListFontsWithInfo`, `XLoadFont`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XFreeFontPath

Name

XFreeFontPath — free the memory allocated by `XGetFontPath`.

Synopsis

```
XFreeFontPath(list)
    char **list;
```

Arguments

list	Specifies an array of strings allocated by `XGetFontPath`.

Description

`XFreeFontPath` frees the data used by the array of pathnames returned by `XGetFont-Path`.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Related Commands

`XCreateFontCursor`, `XFreeFont`, `XFreeFontInfo`, `XFreeFontNames`, `XGet-FontPath`, `XGetFontProperty`, `XListFonts`, `XListFontsWithInfo`, `XLoad-Font`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XFreeGC

Name

XFreeGC — free a graphics context.

Synopsis

```
XFreeGC(display, gc)
    Display *display;
    GC gc;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
gc	Specifies the graphics context to be freed.

Description

`XFreeGC` frees all memory associated with a graphics context, and removes the GC from the server and display hardware.

For more information, see Volume One, Chapter 5, *The Graphics Context*.

Errors

`BadGC`

Related Commands

`DefaultGC`, `XChangeGC`, `XCopyGC`, `XCreateGC`, `XGContextFromGC`, `XSetArcMode`, `XSetBackground`, `XSetClipMask`, `XSetClipOrigin`, `XSetClipRectangles`, `XSetDashes`, `XSetFillRule`, `XSetFillStyle`, `XSetForeground`, `XSet-Function`, `XSetGraphicsExposures`, `XSetLineAttributes`, `XSetPlaneMask`, `XSetState`, `XSetStipple`, `XSetSubwindowMode`, `XSetTSOrigin`.

XFreeModifiermap

Name

XFreeModifiermap — destroy and free a keyboard modifier mapping structure.

Synopsis

```
XFreeModifiermap(modmap)
      XModifierKeymap *modmap;
```

Arguments

modmap	Specifies a pointer to the `XModifierKeymap` structure to be freed.

Description

`XFreeModifiermap` frees an `XModifierKeymap` structure originally allocated by `XNew-ModifierMap` or `XGetModifierMapping`.

For more information, see Volume One, Chapter 9, *The Keyboard and Pointer*.

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;
```

Related Commands

`XChangeKeyboardMapping`, `XDeleteModifiermapEntry`, `XGetKeyboard-Mapping`, `XGetModifierMapping`, `XInsertModifiermapEntry`, `XKeycode-ToKeysym`, `XKeysymToKeycode`, `XKeysymToString`, `XLookupKeysym`, `XLookup-String`, `XNewModifierMap`, `XQueryKeymap`, `XRebindKeySym`, `XRefresh-KeyboardMapping`, `XSetModifierMapping`, `XStringToKeysym`.

XFreePixmap

Name

XFreePixmap — free a pixmap ID.

Synopsis

```
XFreePixmap(display, pixmap)
    Display *display;
    Pixmap pixmap;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
pixmap	Specifies the pixmap whose ID should be freed.

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`

Related Commands

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

XFreeStringList

Name

XFreeStringList — free the in-memory data associated with the specified string list.

Synopsis

```
void XFreeStringList(list)
      char **list;
```

Arguments

list	Specifies the list of strings to be freed.

Availability

Release 4 and later.

Description

`XFreeStringList` releases memory allocated by `XTextPropertyToStringList`.

Related Commands

`XGetTextProperty`, `XSetTextProperty`, `XStringListToTextProperty`, `XTextPropertytoStringList`.

XGContextFromGC

Name

XGContextFromGC — obtain the `GContext` (resource ID) associated with the specified graphics context.

Synopsis

```
GContext XGContextFromGC(gc)
    GC gc;
```

Arguments

gc	Specifies the graphics context of the desired resource ID.

Description

`XGContextFromGC` extracts the resource ID from the GC structure. The GC structure is Xlib's local cache of GC values and contains a field for the `GContext` ID. This function is essentially a macro that accesses this field, since the GC structure is intended to be opaque.

A `GContext` is needed to set a field of the `XVisualInfo` structure prior to calling `XGet-VisualInfo`.

Related Commands

`DefaultGC`, `XChangeGC`, `XCopyGC`, `XCreateGC`, `XFreeGC`, `XSetArcMode`, `XSet-Background`, `XSetClipMask`, `XSetClipOrigin`, `XSetClipRectangles`, `XSet-Dashes`, `XSetFillRule`, `XSetFillStyle`, `XSetForeground`, `XSetFunction`, `XSetGraphicsExposures`, `XSetLineAttributes`, `XSetPlaneMask`, `XSetState`, `XSetStipple`, `XSetSubwindowMode`, `XSetTSOrigin`.

XGeometry

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 connection to an X server; 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 and must be complete.
bwidth	Specifies the border width.
fheight *fwidth*	Specify the font height and width in pixels (increment size).
xadder *yadder*	Specify additional interior padding in pixels needed in the window.
x *y*	Return the user-specified or default coordinates of the window.
width *height*	Return the window dimensions in pixels.

Description

`XGeometry` has been superseded by `XWMGeometry` as of Release 4.

`XGeometry` returns the position and size of a window 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 options and user preferences. These geometry strings are of the form:

```
=<width>x<height>{+-}<xoffset>{+-}<yoffset>
```

The "=" at the beginning of the string is now optional. (Items enclosed in <> are integers, and items enclosed in { } are a set from which one item is to be chosen. Note that the brackets should not appear in the actual string.)

The `XGeometry` return value is a bitmask that 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.

Related Commands

`XParseGeometry`, `XTranslateCoordinates`, `XWMGeometry`.

XGetAtomName

Name

XGetAtomName — get a string name for a property given its atom.

Synopsis

```
char *XGetAtomName(display, atom)
    Display *display;
    Atom atom;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
atom	Specifies the atom whose string name you want returned.

Description

An atom is a number identifying a property. Properties also have a string name. `XGetAtomName` returns the string name that was specified in the original call to `XInternAtom` that returned this atom, or, for predefined atoms, a string version of the symbolic constant without the XA_ is returned. If the specified atom is not defined, `XGetAtomName` returns `NULL`, and generates a `BadAtom` error.

For example, `XGetAtomName` returns `"XA_WM_CLASS"` (a string) when passed the predefined atom `XA_WM_CLASS` (a defined constant).

You should free the resulting string with `XFree` when it is no longer needed.

`XInternAtom` performs the inverse function, returning the atom given the string.

Errors

`BadAtom`

Related Commands

`XChangeProperty`, `XDeleteProperty`, `XGetFontProperty`, `XGetWindowProperty`, `XInternAtom`, `XListProperties`, `XRotateWindowProperties`, `XSetStandardProperties`.

XGetClassHint

Name

XGetClassHint — get the `XA_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 connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window for which the property is desired.
class_hints	Returns the `XClassHints` structure.

Description

`XGetClassHint` obtains the `XA_WM_CLASS` property for the specified window. This property stores the resource class and instance name, that the window manager uses to get any resource settings that may control how the window manager manages the application that set this property. `XGetClassHint` returns a `Status` of zero on failure, nonzero on success.

The `XClassHint` structure returned contains `res_class`, which is the name of the client such as "emacs", and `res_name`, which should be 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]` (after the last /)

To free `res_name` and `res_class` when finished with the strings, use `XFree`.

For more information on using hints, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    char *res_name;
    char *res_class;
} XClassHint;
```

Errors

`BadWindow`

Related Commands

`XAllocClassHint`, `XFetchName`, `XGetIconName`, `XGetIconSizes`, `XGetNormalHints`, `XGetSizeHints`, `XGetTransientForHint`, `XGetWMHints`, `XGetZoomHints`, `XSetClassHint`, `XSetCommand`, `XSetIconName`, `XSetIconSizes`, `XSetNormalHints`, `XSetSizeHints`, `XSetTransientForHint`, `XSetWMHints`, `XSetZoomHints`, `XStoreName`, `XSetWMProperties`, `XSetWMProperties`.

XGetCommand

Name

XGetCommand — get the XA_WM_COMMAND property (command line arguments).

Synopsis

```
Status XGetCommand(display, w, argv_return, argc_return)
    Display *display;
    Window w;
    char ***argv_return;
    int *argc_return;
```

Arguments

display	Specifies a connection to an X server; returned from XOpenDisplay.
w	Specifies the window.
argv_return	Returns the application's argument list.
argc_return	Returns the number of arguments returned.

Description

XGetCommand reads the XA_WM_COMMAND property from the specified window and returns a string list. If the XA_WM_COMMAND property exists, it is of type XA_STRING and format 8. If sufficient memory can be allocated to contain the string list, XGetCommand fills in the *argv_return* and *argc_return* arguments and returns a non-zero status. Otherwise, it returns a zero status. To free the memory allocated to the string list, use XFreeStringList.

Errors

BadWindow

Related Commands

XFetchName, XGetClassHint, XGetIconName, XGetIconSizes, XGetNormal-Hints, XGetSizeHints, XGetTransientForHint, XGetWMHints, XGetZoom-Hints, XSetClassHint, XSetIconName, XSetIconSizes, XSetNormalHints, XSetSizeHints, XSetTransientForHint, XSetWMHints, XSetZoomHints, XStoreName.

Name

XGetDefault — extract an option value from the resource database.

Synopsis

```
char *XGetDefault(display, program, option)
    Display *display;
    char *program;
    char *option;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
program	Specifies the program name to be looked for in the resource database. The program name is usually `argv[0]`, the first argument on the UNIX command line.
option	Specifies the option name or keyword. Lines containing both the *program* name and the *option* name, separated only by a period or asterisk, will be matched.

Description

`XGetDefault` returns a character string containing the user's default value for the specified *program* name and *option* name. `XGetDefault` returns `NULL` if no key can be found that matches *option* and *program*. For a description of the matching rules, see `XrmGetResource`.

The strings returned by `XGetDefault` are owned by Xlib and should not be modified or freed by the client.

Lines in the user's resource database look like this:

```
xterm.foreground:          #c0c0ff
xterm.geometry:            =81x28
xterm.saveLines:           256
xterm.font:                8x13
xterm.keyMapFile:          /usr/black/.keymap
xterm.activeIcon:          on
xmh.header.font            9x15
```

The portion on the left is known as a key; the portion on the right is the value. Upper or lower case is important in keys. The convention is to capitalize only the second and successive words in each option, if any.

Resource specifications are usually loaded into the `XA_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 UNIX-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 and merges the various resource files into Xlib so that subsequent requests are fast. Therefore, changes to the resource files from the program will not be felt until the next invocation of the application.

For more information, see Volume One, Chapter 11, *Managing User Preferences*.

Related Commands

`XAutoRepeatOff`, `XAutoRepeatOn`, `XBell`, `XChangeKeyboardControl`, `XGet-KeyboardControl`, `XGetPointerControl`.

XGetErrorDatabaseText

Name

XGetErrorDatabaseText — obtain error messages from the error database.

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 connection to an X server; 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 UNIX-based system, the error message database is usually */usr/lib/X11/XErrorDB*.

The *name* argument should generally be the name of your application. The *message* argument should indicate which type of error message you want. 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 Xlib.
`XRequestMajor`	The major request protocol number is used for the message argument.

If no string is found in the error database, `XGetErrorDatabaseText` returns the `default_string` that you specify to the buffer. The string in *buffer* will be of length *length*. For more information, see Volume One, Chapter 3, *Basic Window Program*.

Related Commands

`XDisplayName`, `XGetErrorText`, `XSetAfterFunction`, `XSetErrorHandler`, `XSetIOErrorHandler`, `XSynchronize`.

XGetErrorText

Name

XGetErrorText — obtain a description of error code.

Synopsis

```
XGetErrorText(display, code, buffer, length)
      Display *display;
      int code;
      char *buffer;                 /* RETURN */
      int length;
```

Arguments

display	Specifies a connection to an X server; 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 that can be accessed easily.

For more information, see Volume One, Chapter 3, *Basic Window Program*.

Related Commands

`XDisplayName`, `XGetErrorDatabaseText`, `XSetAfterFunction`, `XSetError-Handler`, `XSetIOErrorHandler`, `XSynchronize`.

XGetFontPath

Name

XGetFontPath — get the current font search path.

Synopsis

```
char **XGetFontPath(display, npaths)
    Display *display;
    int *npaths;              /* RETURN number of elements */
```

Arguments

display	Specifies a connection to an X server; 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 for fonts. The data in the font path should be freed when no longer needed.

Related Commands

`XCreateFontCursor`, `XFreeFont`, `XFreeFontInfo`, `XFreeFontNames`, `XFree-FontPath`, `XGetFontProperty`, `XListFonts`, `XListFontsWithInfo`, `XLoad-Font`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XGetFontProperty

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 `False` if the atom was not defined, or `True` 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 Volume One, Chapter 6, *Drawing Graphics and Text*. This set contains the standard properties associated with a font. The predefined font properties are likely but not guaranteed to be present for any given font.

See Volume One, Appendix I, *Logical Font Description Conventions*, for more information on font properties.

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 nonzero size*/
     unsigned default_char;         /* char to print for undefined character */
     int n_properties;              /* how many properties there are */
     XFontProp *properties;         /* pointer to array of additional properties*/
     XCharStruct min_bounds;        /* minimum bounds over all existing char*/
     XCharStruct max_bounds;        /* 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;
```

Related Commands

`XChangeProperty`, `XDeleteProperty`, `XGetAtomName`, `XGetWindowProperty`, `XInternAtom`, `XListProperties`, `XRotateWindowProperties`, `XSetStandard-Properties`.

XGetGCValues

Name

XGetGCValues — obtain components of a given GC from Xlib's GC cache.

Synopsis

```
Status XGetGCValues(display, gc, valuemask, values)
      Display *display;
      GC gc;
      unsigned long valuemask;
      XGCValues *values;            /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
gc	Specifies the GC.
valuemask	Specifies which components in the GC are to be returned in the `values` argument. This argument is the bitwise inclusive OR of one or more of the valid GC component mask bits.
values	Returns the GC values in the specified `XGCValues` structure.

Availability

Release 4 and later.

Description

`XGetGCValues` returns the components specified by valuemask for the specified GC. Note that the clip mask and dash list (represented by the `GCClipMask` and `GCDashList` bits, respectively, in the valuemask) cannot be requested. If the valuemask contains a valid set of GC mask bits (any of those listed in the Structures section with the exception of `GCClipMask` and `GCDashList`) and no error occur, `XGetGCValues` sets the requested components in `values` and returns a nonzero status. Otherwise, it returns a zero status.

For more information, see Volume One, Chapter 5, *The Graphics Context*.

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;                  /* 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)     /* not valid in this call */
    #define GCDashOffset            (1L<<20)
    #define GCDashList              (1L<<21)     /* not valid in this call */
    #define GCArcMode               (1L<<22)
```

Related Commands

XChangeGC, XCopyGC, XCreateGC.

XGetGeometry

Name

XGetGeometry — obtain the current geometry of drawable.

Synopsis

```
Status XGetGeometry(display, drawable, root, x, y,
        width, height, border_width, depth)
    Display *display;
    Drawable drawable;
    Window *root;                       /* RETURN */
    int *x, *y;                         /* RETURN */
    unsigned int *width, *height;       /* RETURN */
    unsigned int *border_width;         /* RETURN */
    unsigned int *depth;                /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable, either a window or a pixmap.
root	Returns the root window ID of the specified window.
x *y*	Return the coordinates of the upper-left pixel of the window's border, relative to its parent's origin. For pixmaps, these coordinates are always zero.
width *height*	Return the dimensions of the drawable. For a window, these return the inside size (not including the border).
border_width	Returns the borderwidth, in pixels, of the window's border, if the drawable is a window. Returns zero if the drawable is a pixmap.
depth	Returns the depth of the pixmap or window (bits per pixel for the object).

Description

This function gets the current geometry of a drawable, plus the ID of the root window of the screen the window is on.

`XGetGeometry` returns a `Status` of zero on failure, or nonzero on success.

Errors

`BadDrawable`

Related Commands

`XConfigureWindow`, `XGetWindowAttributes`, `XMoveResizeWindow`, `XMove-Window`, `XResizeWindow`.

XGetIconName

Name

XGetIconName — get the name to be displayed in an icon.

Synopsis

```
Status XGetIconName(display, w, icon_name)
    Display *display;
    Window w;
    char **icon_name;            /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of 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 a name hasn't been assigned 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` is superseded by `XGetWMIconName` in Release 4. `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 a window's icon when they need to display that icon.

`XGetIconName` returns a nonzero `Status` if it succeeds, and zero if no icon name has been set for the argument window.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadWindow`

Related Commands

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

XGetIconSizes

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 connection to an X server; 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` reads the `XA_WM_ICON_SIZE` property that should be set by the window manager to specify its desired icon sizes. `XGetIconSizes` returns a `Status` of zero if a window manager has not set icon sizes, and a nonzero `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. To free the data allocated in `size_list`, use `XFree`.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

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`

Related Commands

XAllocIconSize, XFetchName, XGetClassHint, XGetIconName, XGetNormal-Hints, XGetSizeHints, XGetTransientForHint, XGetWMHints, XGetZoom-Hints, XSetClassHint, XSetCommand, XSetIconSizes, XSetNormalHints, XSetSizeHints, XSetTransientForHint, XSetWMHints, XSetZoomHints, XStoreName.

XGetImage

Name

XGetImage — place contents of a rectangle from drawable into an image.

Synopsis

```
XImage *XGetImage(display, drawable, x, y, width, height,
         plane_mask, format)
    Display *display;
    Drawable drawable;
    int x, y;
    unsigned int width, height;
    unsigned long plane_mask;
    int format;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable to get the data from.
x *y*	Specify the x and y coordinates of the upper-left corner of the rectangle, relative to the origin of the drawable.
width *height*	Specify the width and height in pixels of the image.
plane_mask	Specifies a plane mask that indicates which planes are represented in the image.
format	Specifies the format for the image. Pass either `XYPixmap` or `ZPixmap`.

Description

`XGetImage` dumps the contents of the specified rectangle, a drawable, into a client-side `XImage` structure, in the format you specify. Depending on which format you pass to the format argument, the function does the following:

- If the format is `XYPixmap`

  Gets only the bit planes you passed to the *plane_mask* argument.

- If the format is `ZPixmap`

  Sets to 0 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. This depth 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, and the `visual` field in the image will be `None`. If `XGetImage` fails, it returns `NULL`. If the drawable is a window, the window must be viewable, and the specified rectangle must not go off the edge of the screen. Otherwise, a `BadMatch` error will occur. If the drawable is a window, the *visual* argument will return the visual specified when the drawable was created.

The returned image will include any visible portions of inferiors or overlapping windows contained in the rectangle. The image will not include the cursor. 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.

The data in the image structure is stored in the server's natural byte- and bit-order.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

`BadDrawable`	
`BadMatch`	See Description above.
`BadValue`	

Related Commands

`ImageByteOrder`, `XAddPixel`, `XCreateImage`, `XDestroyImage`, `XGetPixel`, `XGetSubImage`, `XPutImage`, `XPutPixel`, `XSubImage`.

XGetInputFocus

Name

XGetInputFocus — return the current keyboard focus window.

Synopsis

```
XGetInputFocus(display, focus, revert_to)
    Display *display;
    Window *focus;              /* RETURN */
    int *revert_to;             /* RETURN */
```

Arguments

display	Specifies a connection to an X server; 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 `RevertToNone`. Must not be a window ID.

Description

`XGetInputFocus` returns the current keyboard focus window and the window to which the focus would revert if the focus window became invisible.

`XGetInputFocus` does not report the last focus change time. This is available only from `FocusIn` and `FocusOut` events.

Related Commands

`QLength`, `XAllowEvents`, `XCheckIfEvent`, `XCheckMaskEvent`, `XCheckTyped-Event`, `XCheckTypedWindowEvent`, `XCheckWindowEvent`, `XEventsQueued`, `XGetMotionEvents`, `XIfEvent`, `XMaskEvent`, `XNextEvent`, `XPeekEvent`, `XPeek-IfEvent`, `XPending`, `XPutBackEvent`, `XSelectInput`, `XSendEvent`, `XSetInput-Focus`, `XSynchronize`, `XWindowEvent`.

XGetKeyboardControl

Name

XGetKeyboardControl — obtain a list of the current keyboard preferences.

Synopsis

```
XGetKeyboardControl(display, values)
    Display *display;
    XKeyboardState *values;   /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
values	Returns filled `XKeyboardState` structure.

Description

`XGetKeyboardControl` returns the current control values for the keyboard. For the LEDs (light emitting diodes), the least significant bit of *led_mask* corresponds to LED 1, and each bit that is set to 1 in *led_mask* indicates an LED that is lit. `auto_repeats` is a bit vector; each bit that is set to 1 indicates that auto-repeat is enabled for the corresponding key. The vector is represented as 32 bytes. Byte N (from 0) contains the bits for keys 8N to 8N+7, with the least significant bit in the byte representing key 8N. *global_auto_repeat* is either `AutoRepeatModeOn` or `AutoRepeatModeOff`.

For the ranges of each member of `XKeyboardState`, see the description of `XChangePointerControl`.

For more information, see Volume One, Chapter 9, *The Keyboard and Pointer*.

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;
```

Related Commands

`XAutoRepeatOff`, `XAutoRepeatOn`, `XBell`, `XChangeKeyboardControl`, `XGetDefault`, `XGetPointerControl`.

XGetKeyboardMapping

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 connection to an X server; 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 by `XDisplayKeycodes`, otherwise a `BadValue` error occurs. In addition, the following expression must be less than or equal to `max_keycode` (also returned by `XDisplayKeycodes`) 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 0) for keycode *K* has an index (counting from 0) 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.

For more information, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Errors

`BadValue`	*`first_keycode`* less than *`display`*`->min_keycode`.
	`display``->max_keycode` exceeded.

Related Commands

`XChangeKeyboardMapping`, `XDeleteModifiermapEntry`, `XFreeModifiermap`, `XGetModifierMapping`, `XInsertModifiermapEntry`, `XKeycodeToKeysym`, `XKeysymToKeycode`, `XKeysymToString`, `XLookupKeysym`, `XLookupString`, `XNewModifierMap`, `XQueryKeymap`, `XRebindKeySym`, `XRefreshKeyboard-Mapping`, `XSetModifierMapping`, `XStringToKeysym`.

XGetModifierMapping

Name

XGetModifierMapping — obtain a mapping of modifier keys (Shift, Control, etc.).

Synopsis

```
XModifierKeymap *XGetModifierMapping(display)
    Display *display;
```

Arguments

display	Specifies a connection to an X server; 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 zero's, 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 modifier */
    KeyCode *modifiermap;  /* 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. */
#define ShiftMapIndex       0
#define LockMapIndex        1
#define ControlMapIndex     2
#define Mod1MapIndex        3
#define Mod2MapIndex        4
#define Mod3MapIndex        5
#define Mod4MapIndex        6
#define Mod5MapIndex        7
```

Related Commands

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

XGetMotionEvents

Name

XGetMotionEvents — get events from pointer motion history buffer.

Synopsis

```
XTimeCoord *XGetMotionEvents(display, w, start, stop, nevents)
      Display *display;
      Window w;
      Time start, stop;
      int *nevents;                    /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window whose associated pointer motion events will be returned.
start *stop*	Specify the time interval for 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*.

`XGetMotionEvent` returns `NULL` if the server does not support a motion history buffer (which is common), or if the start time is after the stop time, or if the start time is in the future. A motion history buffer is supported if `XDisplayMotionBufferSize` (display) > `0`. The pointer position at each pointer hardware interrupt is then stored for later retrieval.

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`, since the server does not wait to report future events.

Use `XFree` to free the returned `XTimeCoord` structures when they are no longer needed.

For more information, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Structures

```
typedef struct _XTimeCoord {
      Time time;
      short x, y;
} XTimeCoord;
```

Errors

`BadWindow`

Related Commands

QLength, XAllowEvents, XCheckIfEvent, XCheckMaskEvent, XCheckTyped-Event, XCheckTypedWindowEvent, XCheckWindowEvent, XEventsQueued, XGetInputFocus, XIfEvent, XMaskEvent, XNextEvent, XPeekEvent, XPeek-IfEvent, XPending, XPutBackEvent, XSelectInput, XSendEvent, XSetInput-Focus, XSynchronize, XWindowEvent.

XGetNormalHints

Name

XGetNormalHints — get the size hints property of a 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 connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window to be queried.
hints	Returns the sizing hints for the window in its normal state.

Description

`XGetNormalHints` has been superseded by `XGetWMNormalHints` as of Release 4, because new interclient communication conventions are now standard.

`XGetNormalHints` returns the size hints for a window in its normal state by reading the `XA_WM_NORMAL_HINTS` property. This function is normally used only by a window manager. It returns a nonzero `Status` if it succeeds, and zero if it fails (e.g., the application specified no normal size hints for this window.)

For more information on using hints, see Volume One, Chapter 10, *Interclient Communication*.

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

Related Commands

XFetchName, XGetClassHint, XGetIconName, XGetIconSizes, XGetSize-Hints, XGetTransientForHint, XGetWMHints, XGetZoomHints, XSetClass-Hint, XSetCommand, XSetIconName, XSetIconSizes, XSetNormalHints, XSet-SizeHints, XSetTransientForHint, XSetWMHints, XSetZoomHints, XStore-Name.

XGetPixel

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 of the pixel whose value is to be returned.

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 the clients bit- and byte-order. The x and y coordinates must be contained in the image.

For more information, see Volume One, Chapter 6, *Drawing Graphics and Text*.

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 scan line 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;             /* accelerator 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;
```

Related Commands

`ImageByteOrder`, `XAddPixel`, `XCreateImage`, `XDestroyImage`, `XGetImage`, `XGetSubImage`, `XPutImage`, `XPutPixel`, `XSubImage`.

XGetPointerControl

Name

XGetPointerControl — get the current pointer preferences.

Synopsis

```
XGetPointerControl(display, accel_numerator, accel_denominator,
        threshold)
    Display *display;
    int *accel_numerator, *accel_denominator;   /* RETURN */
    int *threshold;                              /* RETURN */
```

Arguments

display	Specifies a connection to an X server; 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 divided by *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*.

Related Commands

`XChangeActivePointerGrab`, `XChangePointerControl`, `XGetPointer-Mapping`, `XGrabPointer`, `XQueryPointer`, `XSetPointerMapping`, `XUngrab-Pointer`, `XWarpPointer`.

XGetPointerMapping

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 connection to an X server; 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. Information is returned in both the arguments and the function's return value. *map* is an array of the numbers of the buttons as they are currently mapped. Elements of the list are indexed starting from 1. 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 return value of the function is the actual number of elements in the pointer list, which may be greater or less than *nmap*.

Related Commands

`XChangeActivePointerGrab`, `XChangePointerControl`, `XGetPointer-Control`, `XGrabPointer`, `XQueryPointer`, `XSetPointerMapping`, `XUngrab-Pointer`, `XWarpPointer`.

XGetRGBColormaps

Name

XGetRGBColormaps — obtain the `XStandardColormap` structure associated with the specified property.

Synopsis

```
Status XGetRGBColormaps(display, w, std_colormap, count,
              property)
        Display *display;
        Window w;
        XStandardColormap **std_colormap;     /* RETURN */
        int *count;                           /* RETURN */
        Atom property;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window.
std_colormap	Returns the `XStandardColormap` structure.
count	Returns the number of colormaps.
property	Specifies the property name.

Availability

Release 4 and later.

Description

`XGetRGBColormaps` returns the RGB colormap definitions stored in the specified property on the named window. If the property exists, is of type `RGB_COLOR_MAP`, is of format 32, and is long enough to contain a colormap definition, `XGetRGBColormaps` allocates and fills in space for the returned colormaps, and returns a non-zero status. Otherwise, none of the fields are set, and `XGetRGBColormaps` returns a zero status. If the visualid field is not present, `XGetRGBColormaps` assumes the default visual for the screen on which the window is located; if the killid field is not present, it is assumed to have a value of `None`, which indicates that the resources cannot be released. Note that it is the caller's responsibility to honor the ICCCM restriction that only `RGB_DEFAULT_MAP` contain more than one definition.

`XGetRGBColormaps` supersedes `XGetStandardColormap`.

For more information, see Volume One, Chapter 7, *Color*.

Structures

```
typedef struct {
    Colormap colormap;
    unsigned long red_max;
    unsigned long red_mult;
    unsigned long green_max;
```

```
        unsigned long green_mult;
        unsigned long blue_max;
        unsigned long blue_mult;
        unsigned long base_pixel;
        VisualID visualid;                /* added by ICCCM version 1 */
        XID killid;                       /* added by ICCCM version 1 */
    } XStandardColormap;
```

Errors

BadAtom
BadWindow

Related Commands

XAllocStandardColormap, XSetRGBColormaps.

XGetScreenSaver

Name

XGetScreenSaver — get the 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 connection to an X server; returned from `XOpenDisplay`.
timeout	Returns the idle time, in seconds, until the screen saver turns on.
interval	Returns the interval between screen changes, 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 zero indicates that the screen saver is disabled.

If the server-dependent screen saver method supports periodic change, *interval* serves as a hint about the length of the change period, and zero serves as a hint that no periodic change will be made. An *interval* of zero indicates that random pattern motion is disabled.

For more information on the screen saver, see Volume One, Chapter 13, *Other Programming Techniques*.

Related Commands

`XActivateScreenSaver`, `XForceScreenSaver`, `XResetScreenSaver`, `XSetScreenSaver`.

XGetSelectionOwner

Name

XGetSelectionOwner — return the owner of a selection.

Synopsis

```
Window XGetSelectionOwner(display, selection)
      Display *display;
      Atom selection;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
selection	Specifies the selection 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`.

For more information on selections, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadAtom`

Related Commands

`XConvertSelection`, `XSetSelectionOwner`.

XGetSizeHints

Name

XGetSizeHints — read any property of type XA_SIZE_HINTS.

Synopsis

```
Status XGetSizeHints(display, w, hints, property)
    Display *display;
    Window w;
    XSizeHints *hints;          /* RETURN */
    Atom property;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window for which size hints will be returned.
hints	Returns the size hints structure.
property	Specifies a property atom of type `XA_WM_SIZE_HINTS`. May be `XA_WM_NORMAL_HINTS`, `XA_WM_ZOOM_HINTS` (in Release 3), or a property defined by an application.

Description

`XGetSizeHints` has been superseded by `XGetWMSizeHints` as of Release 4, because the interclient communication conventions are now standard.

`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 `XA_WM_SIZE_HINTS`; thus, it is useful if other properties of that type get defined. This function is used almost exclusively by window managers.

`XGetSizeHints` returns a nonzero `Status` if a size hint was defined, and zero otherwise.

For more information on using hints, see Volume One, Chapter 10, *Interclient Communication*.

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

Related Commands

XFetchName, XGetClassHint, XGetIconName, XGetIconSizes, XGetNormal-Hints, XGetTransientForHint, XGetWMHints, XGetZoomHints, XSetClass-Hint, XSetCommand, XSetIconName, XSetIconSizes, XSetNormalHints, XSet-SizeHints, XSetTransientForHint, XSetWMHints, XSetZoomHints, XStore-Name.

XGetStandardColormap

Name

XGetStandardColormap — get the standard colormap property.

Synopsis

```
Status XGetStandardColormap(display, w, cmap_info, property)
    Display *display;
    Window w;
    XStandardColormap *cmap_info;/* RETURN */
    Atom property;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window on which the property is set. This is normally the root window.
cmap_info	Returns the filled colormap information structure.
property	Specifies the atom indicating the type of standard colormap desired. The predefined standard colormap atoms are `XA_RGB_BEST_MAP`, `XA_RGB_RED_MAP`, `XA_RGB_GREEN_MAP`, `XA_RGB_BLUE_MAP`, `XA_RGB_DEFAULT_MAP`, and `XA_RGB_GRAY_MAP`.

Description

`XGetStandardColormap` is superseded by `XGetWMColormap` in Release 4.

`XGetStandardColormap` gets a property on the root window that describes a standard colormap.

This call does not install 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 design of colormap and the ID of the colormap if some other client has already created it. The application can otherwise attempt to create a virtual colormap of the appropriate type, and allocate its entries according to the information in the `XStandardColormap` structure. Installing the 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.

If the server or another client has already created a standard colormap of this type, then its ID will be returned in the `colormap` member of the `XStandardColormap` structure. Some servers and window managers, particular on high-performance workstations, will create some or all of the standard colormaps so they can be quickly installed when needed by applications.

An application should go through the standard colormap creation process only if it needs the special qualities of the standard colormaps. For one, they allow the application to convert RGB values into pixel values quickly because the mapping is predictable. Given an `XStandard-Colormap` structure for an `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.

`XGetStandardColormap` returns zero if it fails, or nonzero if it succeeds.

See Volume One, Chapter 7, *Color*, 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;
    /* new fields here in R4 */
} XStandardColormap;
```

Errors

`BadAtom`
`BadWindow`

Related Commands

`DefaultColormap`, `DisplayCells`, `XCopyColormapAndFree`, `XCreate-Colormap`, `XFreeColormap`, `XInstallColormap`, `XListInstalledColormaps`, `XSetStandardColormap`, `XSetWindowColormap`, `XUninstallColormap`.

XGetSubImage

Name

XGetSubImage — copy a rectangle in drawable to a location within the pre-existing image.

Synopsis

```
XImage *XGetSubImage(display, drawable, x, y, width, height,
        plane_mask, format, dest_image, dest_x, dest_y)
    Display *display;
    Drawable drawable;
    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 connection to an X server; returned from `XOpenDisplay`.
drawable	Specifies the drawable from which the rectangle is to be copied.
x *y*	Specify the x and y coordinates of the upper-left corner of the rectangle, relative to the origin of the drawable.
width *height*	Specify the width and height in pixels of the subimage taken.
plane_mask	Specifies which planes of the drawable are transferred to the 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's upper left corner, relative to the image's origin.

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 `BadMatch` 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, and it must be the case that, if there were no inferiors or overlapping windows, the specified rectangle of the window would be fully visible on the screen. 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`.

For more information on images, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

`BadDrawable`	
`BadMatch`	Depth of *`dest_image`* is not the same as depth of *`drawable`*.
`BadValue`	

Related Commands

`ImageByteOrder`, `XAddPixel`, `XCreateImage`, `XDestroyImage`, `XGetImage`, `XGetPixel`, `XPutImage`, `XPutPixel`, `XSubImage`.

XGetTextProperty

Name

XGetTextProperty — read one of a window's text properties.

Synopsis

```
Status XGetTextProperty(display, w, text_prop, property)
      Display *display;
      Window w;
      XTextProperty *text_prop;      /* RETURN */
      Atom property;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window.
text_prop	Returns the `XTextProperty` structure.
property	Specifies the property name.

Availability

Release 4 and later.

Description

`XGetTextProperty` reads the specified property from the window and stores the data in the returned `XTextProperty` structure. It stores the data in the `value` field, the type of the data in the `encoding` field, the format of the data in the `format` field, and the number of items of data in the `nitems` field. The particular interpretation of the property's encoding and data as "text" is left to the calling application. If the specified property does not exist on the window, `XGetTextProperty` sets the value field to `NULL`, the encoding field to `None`, the format field to zero, and the `nitems` field to zero.

If it was able to set these files in the `XTextProperty` structure, `XGetTextProperty` returns a non-zero status; otherwise, it returns a zero status.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    unsigned char *value;        /* same as Property routines */
    Atom encoding;               /* prop type */
    int format;                  /* prop data format: 8, 16, or 32 */
    unsigned long nitems;        /* number of data items in value */
} XTextProperty;
```

Errors

```
BadAtom
BadWindow
```

Related Commands

`XFreeStringList`, `XSetTextProperty`, `XStringListToTextProperty`, `XText-PropertytoStringList`.

XGetTransientForHint

Name

XGetTransientForHint — get the `XA_WM_TRANSIENT_FOR` property of a window.

Synopsis

```
Status XGetTransientForHint(display, w, prop_window)
      Display *display;
      Window w;
      Window *prop_window;        /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window to be queried.
prop_window	Returns the window contained in the `XA_WM_TRANSIENT_FOR` property of the specified window.

Description

`XGetTransientForHint` obtains the `XA_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 dialog boxes. The window returned is the main window to which this popup window is related. This lets the window manager decorate the popup window appropriately.

`XGetTransientForHint` returns a `Status` of zero on failure, and nonzero on success.

For more information on using hints, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadWindow`

Related Commands

`XFetchName`, `XGetClassHint`, `XGetIconName`, `XGetIconSizes`, `XGetNormal-Hints`, `XGetSizeHints`, `XGetWMHints`, `XGetZoomHints`, `XSetClassHint`, `XSet-Command`, `XSetIconName`, `XSetIconSizes`, `XSetNormalHints`, `XSetSizeHints`, `XSetTransientForHint`, `XSetWMHints`, `XSetZoomHints`, `XStoreName`.

XGetVisualInfo

Name

XGetVisualInfo — find the visual information structures that match the specified 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 connection to an X server; 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 describe visuals supported by the server and 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`.

For more information, see Volume One, Chapter 7, *Color*.

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
```

Related Commands

DefaultVisual, XVisualIDFromVisual, XMatchVisualInfo, XListDepths.

XGetWMIconName

Name

XGetWMIconName — read a window's XA_WM_ICON_NAME property.

Synopsis

```
Status XGetWMIconName(display, w, text_prop)
      Display *display;
      Window w;
      XTextProperty *text_prop;/* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window.
text_prop	Returns the `XTextProperty` structure.

Availability

Release 4 and later.

Description

`XGetWMIconName` performs an `XGetTextProperty` on the `XA_WM_ICON_NAME` property of the specified window. `XGetWMIconName` supersedes `XGetIconName`.

This function is primarily used by window managers to get the name to be written in a window's icon when they need to display that icon.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    unsigned char *value;          /* same as Property routines */
    Atom encoding;                 /* prop type */
    int format;                    /* prop data format: 8, 16, or 32 */
    unsigned long nitems;          /* number of data items in value */
} XTextProperty;
```

Related Commands

`XGetWMName`, `XSetWMIconName`, `XSetWMName`, `XSetWMProperties`.

XGetWMName

Name

XGetWMName — read a window's XA_WM_NAME property.

Synopsis

```
Status XGetWMName(display, w, text_prop)
      Display *display;
      Window w;
      XTextProperty *text_prop;/* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from XOpenDisplay.
w	Specifies the window.
text_prop	Returns the XTextProperty structure.

Availability

Release 4 and later.

Description

XGetWMName performs an XGetTextProperty on the XA_WM_NAME property of the specified window. XGetWMName supersedes XFetchName.

XGetWMName returns nonzero if it succeeds, and zero if the property has not been set for the argument window.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    unsigned char *value;       /* same as Property routines */
    Atom encoding;              /* prop type */
    int format;                 /* prop data format: 8, 16, or 32 */
    unsigned long nitems;       /* number of data items in value */
} XTextProperty;
```

Related Commands

XGetWMIconName, XSetWMIconName, XSetWMName, XSetWMProperties.

XGetWMNormalHints

Name

XGetWMNormalHints — read a window's `XA_WM_NORMAL_HINTS` property.

Synopsis

```
Status XGetWMNormalHints(display, w, hints, supplied)
      Display *display;
      Window w;
      XSizeHints *hints;/* RETURN */
      long *supplied;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window.
hints	Returns the size hints for the window in its normal state.
supplied	Returns the hints that were supplied by the user.

Availability

Release 4 and later.

Description

`XGetWMNormalHints` returns the size hints stored in the `XA_WM_NORMAL_HINTS` property on the specified window. If the property is of type `XA_WM_SIZE_HINTS`, of format 32, and is long enough to contain either an old (pre-ICCCM) or new size hints structure, `XGetWMNormalHints` sets the various fields of the `XSizeHints` structure, sets the *supplied* argument to the list of fields that were supplied by the user (whether or not they contained defined values) and returns a non-zero status. `XGetWMNormalHints` returns a zero status if the application specified no normal size hints for this window.

`XGetWMNormalHints` supersedes `XGetNormalHints`.

If `XGetWMNormalHints` returns successfully and a pre-ICCCM size hints property is read, the *supplied* argument will contain the following bits:

```
(USPosition|USSize|PPosition|PSize|PMinSize| PMaxSize|PResizeInc|PAspect)
```

If the property is large enough to contain the base size and window gravity fields as well, the `supplied` argument will also contain the following bits:

```
(PBaseSize|PWinGravity)
```

This function is normally used only by a window manager.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    long flags;     /* marks which fields in this structure are defined */
    int x, y;       /* obsolete for new window mgrs, but clients */
```

```
        int width, height;   /* should set so old wm's don't mess up */
        int min_width, min_height;
        int max_width, max_height;
        int width_inc, height_inc;
        struct {
                int x;  /* numerator */
                int y;  /* denominator */
        } min_aspect, max_aspect;
        int base_width, base_height;      /* added by ICCCM version 1 */
        int win_gravity;                  /* added by ICCCM
    version 1 */
    } XSizeHints;
```

Errors

BadWindow

Related Commands

XAllocSizeHints, XGetWMSizeHints, XSetWMNormalHints, XSet-WMProperties, XSetWMSizeHints.

XGetWMSizeHints

Name

XGetWMSizeHints — read a window's `XA_WM_SIZE_HINTS` property.

Synopsis

```
Status XGetWMSizeHints(display, w, hints, supplied, property)
      Display *display;
      Window w;
      XSizeHints *hints;              /* RETURN */
      long *supplied;                 /*RETURN */
      Atom property;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window.
hints	Returns the `XSizeHints` structure.
supplied	Returns the hints that were supplied by the user.
property	Specifies the property name.

Availability

Release 4 and later.

Description

`XGetWMSizeHints` returns the size hints stored in the specified property on the named window. If the property is of type `XA_WM_SIZE_HINTS`, of format 32, and is long enough to contain either an old (pre-ICCCM) or new size hints structure, `XGetWMSizeHints` sets the various fields of the `XSizeHints` structure, sets the *supplied* argument to the list of fields that were supplied by the user (whether or not they contained defined values), and returns a non-zero status. If the hint was not set, it returns a zero status. To get a window's normal size hints, you can use the `XGetWMNormalHints` function instead.

`XGetWMSizeHints` supersedes `XGetSizeHints`.

If `XGetWMSizeHints` returns successfully and a pre-ICCCM size hints property is read, the *supplied* argument will contain the following bits:

```
(USPosition|USSize|PPosition|PSize|PMinSize| PMaxSize|PResizeInc|PAspect)
```

If the property is large enough to contain the base size and window gravity fields as well, the `supplied` argument will also contain the following bits:

```
(PBaseSize|PWinGravity)
```

This function is used almost exclusively by window managers.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Structures

```
typedef struct {
    long flags;      /* marks which fields in this structure are defined */
    int x, y;        /* obsolete for new window mgrs, but clients */
    int width, height;    /* should set so old wm's don't mess up */
    int min_width, min_height;
    int max_width, max_height;
    int width_inc, height_inc;
    struct {
            int x;   /* numerator */
            int y;   /* denominator */
    } min_aspect, max_aspect;
    int base_width, base_height;            /* added by ICCCM version 1 */
    int win_gravity;                        /* added by ICCCM version 1 */
} XSizeHints;
```

Errors

BadAtom
BadWindow

Related Commands

XAllocSizeHints, XGetWMNormalHints, XSetWMNormalHints, XSetWMSize-Hints.

XGetWindowAttributes

Name

XGetWindowAttributes — obtain the current attributes of window.

Synopsis

```
Status XGetWindowAttributes(display, w, window_attributes)
    Display *display;
    Window w;
    XWindowAttributes *window_attributes; /* RETURN */
```

Arguments

display	Specifies a connection to an X server; 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`.

`XGetWindowAttributes` returns a `Status` of zero on failure, or nonzero on success. However, it will only return zero if you have defined an error handler that does not exit, using `XSetErrorHandler`. The default error handler exits, and therefore `XGetWindow-Attributes` never gets a chance to return. (This is relevant only if you are writing a window manager or other application that deals with windows that might have been destroyed.)

The following list briefly describes each member of the `XWindowAttributes` structure. For more information, see Volume One, Chapter 4, *Window Attributes*.

`x, y`	The current position of the upper-left pixel of the window's border, relative to the origin of its parent.
`width, height`	The current dimensions in pixels of this window.
`border_width`	The current border width of the 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 `Input-Only`.
`bit_gravity`	The new position for existing contents after resize. One of the constants `ForgetGravity`, `StaticGravity`, or `CenterGravity`, or one of the compass constants (`NorthWestGravity`, `NorthGravity`, etc.).

`win_gravity`	The new position for this window after its parent is resized. 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 being 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 such as menus. Either `True` or `False`.
`screen`	A pointer to the `Screen` structure for the screen containing this window.

Errors

`BadWindow`

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;
```

Related Commands

XChangeWindowAttributes, XGetGeometry, XSetWindowBackground, XSet-WindowBackgroundPixmap, XSetWindowBorder, XSetWindowBorderPixmap.

XGetWindowProperty

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_for-
        mat, 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 */
    unsigned long *bytes_after;     /* RETURN */
    unsigned char **prop;           /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window whose atom type and property format you want to obtain.
property	Specifies the atom of the desired property.
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 after being read and a `PropertyNotify` event is generated on the window.
req_type	Specifies an atom describing the desired format of the data. 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 `NULL`. This byte is not counted in *nitems*.

Description

`XGetWindowProperty` gets the value of a property if it is the desired type. `XGetWindow-Property` sets the return arguments acccording to the following rules:

- If the specified property does not exist for the specified window, then: *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*, then: *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, then: *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.

When `XGetWindowProperty` executes successfully, it returns `Success`. The `Success` return value and the undocumented value returned on failure are the opposite of all other routines that return `int` or `Status`. The value of `Success` is undocumented, but is zero (0) in the current sample implementation from MIT. The failure value, also undocumented, is currently one (1). Therefore, comparing either value to `True` or `False`, or using the syntax "`if (!XGetWindowProperty(...))`" is not allowed.

To free the resulting data, use `XFree`.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadAtom`	
`BadValue`	Value of *`long_offset`* caused `L` to be negative above.
`BadWindow`	

Related Commands

`XChangeProperty`, `XGetAtomName`, `XGetFontProperty`, `XListProperties`, `XRotateWindowProperties`, `XSetStandardProperties`.

XGetWMHints

Name

XGetWMHints — read the window manager hints property.

Synopsis

```
XWMHints *XGetWMHints(display, w)
      Display *display;
      Window w;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window to be queried.

Description

This function is primarily for window managers. `XGetWMHints` returns `NULL` if no `XA_WM_HINTS` property was set on window *w*, and returns a pointer to an `XWMHints` structure if it succeeds. Programs must free the space used for that structure by calling `XFree`.

For more information on using hints, see Volume One, Chapter 10, *Interclient Communication*.

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`

Related Commands

`XAllocWMHints`, `XFetchName`, `XGetClassHint`, `XGetIconName`, `XGetIcon-Sizes`, `XGetNormalHints`, `XGetSizeHints`, `XGetTransientForHint`, `XGet-ZoomHints`, `XSetClassHint`, `XSetCommand`, `XSetIconName`, `XSetIconSizes`, `XSetNormalHints`, `XSetSizeHints`, `XSetTransientForHint`, `XSetWMHints`, `XSetZoomHints`, `XStoreName`, `XSetWMProperties`.

XGetZoomHints

Name

XGetZoomHints — read the size hints property of a zoomed window.

Synopsis

```
Status XGetZoomHints(display, w, zhints)
      Display *display;
      Window w;
      XSizeHints *zhints;           /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of the window to be queried.
zhints	Returns a pointer to the zoom hints.

Description

`XGetZoomHints` is obsolete beginning in Release 4, because zoom hints are no longer defined in the ICCCM.

`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 `XA_WM_ZOOM_HINTS` property. It returns a nonzero `Status` if it succeeds, and zero if the application did not specify zoom size hints for this window.

For more information on using hints, see Volume One, Chapter 10, *Interclient Communication*.

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

Related Commands

XFetchName, XGetClassHint, XGetIconName, XGetIconSizes, XGetNormal-Hints, XGetSizeHints, XGetTransientForHint, XGetWMHints, XSetClass-Hint, XSetCommand, XSetIconName, XSetIconSizes, XSetNormalHints, XSet-SizeHints, XSetTransientForHint, XSetWMHints, XSetZoomHints, XStore-Name.

XGrabButton

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 connection to an X server; 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 ID of the window you want to the grab to occur in.
owner_events	Specifies a boolean value of either `True` or `False`. See Description below.
event_mask	Specifies the event mask to take effect during the grab. This mask is the bitwise OR of one or more of the event masks listed on the reference page for `XSelectInput`.
pointer_mode	Controls processing of pointer events during the grab. Pass one of these constants: `GrabModeSync` or `GrabModeAsync`.
keyboard_mode	Controls processing of keyboard events during the grab. Pass one of these constants: `GrabModeSync` or `GrabModeAsync`.
confine_to	Specifies the ID of 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`, in which case the existing cursor is used.

Description

`XGrabButton` establishes a passive grab, such that an active grab may take place when the specified key/button combination is pressed in the specified window. 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 `XGrabPointer`, 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).

`XGrabButton` overrides all previous passive grabs by the same client on the same key/button combination on the same window, but has no effect on an active grab. The request fails if some other client has already issued an `XGrabButton` 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 *owner_events* argument specifies whether the grab window should receive all events (`False`) or whether the grabbing application should receive all events normally (`True`).

The *pointer_mode* and *keyboard_mode* control the processing of events during the grab. If either is `GrabModeSync`, events for that device are not sent from the server to Xlib until `XAllowEvents` is called to release the events. If either is `GrabModeAsync`, events for that device are sent normally.

An automatic grab takes place between a `ButtonPress` event and the corresponding `ButtonRelease` event, so this call is not necessary in some of the most common situations. But this call is necessary for certain styles of menus.

For more information on grabbing, see Volume One, Chapter 9, *The Keyboard and Pointer*.

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.
BadCursor	
BadValue	
BadWindow	

Related Commands

XChangeActivePointerGrab, XGrabKey, XGrabKeyboard, XGrabPointer, XGrabServer, XUngrabButton, XUngrabKey, XUngrabKeyboard, XUngrab-Pointer, XUngrabServer.

XGrabKey

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 connection to an X server; 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 in which the specified key combination will initiate an active grab.
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 processing of pointer events during the grab. Pass one of these constants: `GrabModeSync` or `GrabModeAsync`.
keyboard_mode	Controls processing of keyboard events during the grab. 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 may be grabbed, and all keyboard events sent to this application. More formally, once an XGrabKey call has been issued on a particular key/button combination:

- IF the keyboard is not already actively grabbed,
- AND the specified key, which itself can be a modifier key, is logically pressed when the specified modifier keys are logically down,
- AND no other keys or modifier keys are logically 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 sent from the server to Xlib until `XAllowEvents` is called to send the events. If either is `GrabModeAsync`, events for that device are sent normally.

For more information on grabbing, see Volume One, Chapter 9, *The Keyboard and Pointer*.

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` as returned by `XDisplayKeycodes`.
`BadWindow`	

Related Commands

`XChangeActivePointerGrab`, `XGrabButton`, `XGrabKeyboard`, `XGrabPointer`, `XGrabServer`, `XUngrabButton`, `XUngrabKey`, `XUngrabKeyboard`, `XUngrab-Pointer`, `XUngrabServer`.

XGrabKeyboard

Name

XGrabKeyboard — grab the keyboard.

Synopsis

```
int XGrabKeyboard(display, grab_window, owner_events,
        pointer_mode, keyboard_mode, time)
    Display *display;
    Window grab_window;
    Bool owner_events;
    int pointer_mode, keyboard_mode;
    Time time;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
grab_window	Specifies the ID of the window that requires continuous keyboard input.
owner_events	Specifies a boolean value of either `True` or `False`. See Description below.
pointer_mode	Controls processing of pointer events during the grab. Pass either `GrabModeSync` or `GrabModeAsync`.
keyboard_mode	Controls processing of keyboard events during the grab. 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. 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 by the server until a releasing XAllowEvents request occurs or until the keyboard grab is released as described above.

If the grab is successful, XGrabKeyboard returns the constant GrabSuccess. XGrab-Keyboard 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 Current-Time replaced by the current X server time.

For more information on grabbing, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Errors

BadValue
BadWindow

Related Commands

XChangeActivePointerGrab, XGrabButton, XGrabKey, XGrabPointer, XGrab-Server, XUngrabButton, XUngrabKey, XUngrabKeyboard, XUngrabPointer, XUngrabServer.

XGrabPointer

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 connection to an X server; returned from `XOpenDisplay`.
grab_window	Specifies the ID of the window that should 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 (pass `False`).
event_mask	Specifies the event mask symbols that can be ORed together. Only events selected by this mask, plus `ButtonPress` and `ButtonRelease`, will be delivered during the grab. 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 ID of the window to confine the pointer. One option is `None`, in which case the pointer is not confined to any window.
cursor	Specifies the ID of 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. Further pointer events are only reported to the grabbing client until `XUngrabPointer` is called.

`event_mask` is always augmented to include `ButtonPressMask` and `ButtonRelease-Mask`. If *`owner_events`* is `False`, all generated pointer events are reported 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 processing of pointer events during the grab, 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 by the server 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 which 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. When the pointer is outside `grab_window`, the cursor for *`grab_window`* is displayed.

If a *`confine_to`* window is specified, then the pointer will be restricted to 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 action is most important in this case.

`XGrabPointer` generates `EnterNotify` and `LeaveNotify` events.

If the grab is successful, it returns the constant `GrabSuccess`. The `XGrabPointer` function fails under the following conditions, with the following return values:

- If *`grab_window`* or *`confine_to`* window is not viewable, or if the `confine_to` window is completely off the screen, `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.)

For more information on grabbing, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Errors

`BadCursor`
`BadValue`
`BadWindow`

Related Commands

`XChangeActivePointerGrab`, `XGrabButton`, `XGrabKey`, `XGrabKeyboard`, `XGrabServer`, `XUngrabButton`, `XUngrabKey`, `XUngrabKeyboard`, `XUngrab-Pointer`, `XUngrabServer`.

XGrabServer

Name

XGrabServer — grab the server.

Synopsis

```
XGrabServer(display)
    Display *display;
```

Arguments

display	Specifies a connection to an X server; 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.

Related Commands

`XChangeActivePointerGrab`, `XGrabButton`, `XGrabKey`, `XGrabKeyboard`, `XGrabPointer`, `XUngrabButton`, `XUngrabKey`, `XUngrabKeyboard`, `XUngrabPointer`, `XUngrabServer`.

XIconifyWindow

Name

XIconifyWindow — request that a top-level window be iconified.

Synopsis

```
Status XIconifyWindow(display, w, screen_number)
      Display *display;
      Window w;
      int screen_number;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the window.
screen_number	Specifies the appropriate screen number on the server.

Availability

Release 4 and later.

Description

`XIconifyWindow` sends a `WM_CHANGE_STATE` `ClientMessage` event with a format of 32 and a first data element of `IconicState` (as described in Section 4.1.4 of the *Inter-Client Communication Conventions Manual* in Volume Zero, *X Protocol Reference Manual*), to the root window of the specified screen. Window managers may elect to receive this message and, if the window is in its normal state, may treat it as a request to change the window's state from normal to iconic. If the `WM_CHANGE_STATE` property cannot be interned, `XIconifyWindow` does not send a message and returns a zero status. It returns a nonzero status if the client message is sent successfully; otherwise, it returns a zero status.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadWindow`

Related Commands

`XReconfigureWindow`, `XWithdrawWindow`.

XIfEvent

Name

XIfEvent — wait for event matched in predicate procedure.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 one meets 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 once for each event on the queue until a match is made, and each time an event is added to the queue, with the arguments *display*, *event*, and *arg*.

If no matching events exist on the queue, `XIfEvent` flushes the request buffer and waits for an appropriate event to arrive. Use `XCheckIfEvent` if you don't want to wait for an event.

For more information, see Volume One, Chapter 8, *Events*.

Related Commands

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

XInsertModifiermapEntry

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
```

Related Commands

XDeleteModifiermapEntry, XFreeModifiermap, XGetKeyboardMapping, XGetModifierMapping, XKeycodeToKeysym, XKeysymToKeycode, XKeysymTo-String, XLookupKeysym, XLookupString, XNewModifierMap, XQueryKeymap, XRebindKeySym, XRefreshKeyboardMapping, XSetModifierMapping, XStringToKeysym.

XInstallColormap

Name

XInstallColormap — install a colormap.

Synopsis

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

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the colormap to install.

Description

`XInstallColormap` installs a virtual colormap into a hardware company. 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-performance workstations, `min_maps` is likely to be 1.

If the hardware colormap is immutable, and therefore installing any colormap is impossible, `XInstallColormap` will work but not do anything.

For more information, see Volume One, Chapter 7, *Color*.

Errors

`BadColormap`

Related Commands

DefaultColormap, DisplayCells, XCopyColormapAndFree, XCreate-Colormap, XFreeColormap, XGetStandardColormap, XListInstalled-Colormaps, XSetStandardColormap, XSetWindowColormap, XUninstall-Colormap.

XInternAtom

Name

XInternAtom — return an atom for a given property name string.

Synopsis

```
Atom XInternAtom(display, property_name, only_if_exists)
     Display *display;
     char *property_name;
     Bool only_if_exists;
```

Arguments

display Specifies a connection to an X server; returned from `XOpenDisplay`.

property_name
Specifies the string name of the property for which you want the atom. Upper or lower case is important. The string should be in ISO LATIN-1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.

only_if_exists
Specifies a boolean value: if no such *property_name* exists `XInternAtom` will return `None` if this argument is set to `True` or will create the atom if it is set to `False`.

Description

`XInternAtom` returns the atom identifier corresponding to string *property_name*.

If the atom does not exist, then `XInternAtom` either returns `None` (if *only_if_exists* is `True`) or creates the atom and returns its ID (if *only_if_exists* is `False`).

The string name should be a null-terminated. Case matters: the strings "thing," "Thing," and "thinG" all designate different atoms.

The atom will remain defined even after the client that defined it has exited. It will become undefined only when the last connection to the X server closes. Therefore, the number of atoms interned should be kept to a minimum.

This function is the opposite of `XGetAtomName`, which returns the atom name when given an atom ID.

Predefined atoms require no call to `XInternAtom`. Predefined atoms are defined in *<X11/Xatom.h>* and begin with the prefix "XA_". Predefined atoms are the only ones that do not require a call to `XInternAtom`.

Errors

`BadAlloc`
`BadValue`

Related Commands

`XChangeProperty`, `XDeleteProperty`, `XGetAtomName`, `XGetFontProperty`, `XGetWindowProperty`, `XListProperties`, `XRotateWindowProperties`, `XSetStandardProperties`.

XIntersectRegion

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 region that is the intersection of two regions.

Structures

`Region` is a pointer to an opaque structure type.

Related Commands

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

XKeycodeToKeysym

Name

XKeycodeToKeysym — convert a keycode to a keysym.

Synopsis

```
KeySym XKeycodeToKeysym(display, keycode, index)
    Display *display;
    KeyCode keycode;
    int index;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
keycode	Specifies the keycode.
index	Specifies which keysym in the list for the keycode to return.

Description

`XKeycodeToKeysym` returns one of the keysyms defined for the specified *keycode*. `XKeycodeToKeysym` uses internal Xlib tables. *index* specifies which keysym in the array of keysyms corresponding to a keycode should be returned. If no symbol is defined, `XKeycodeToKeysym` returns `NoSymbol`.

Related Commands

`IsCursorKey`, `IsFunctionKey`, `IsKeypadKey`, `IsMiscFunctionKey`, `IsModifierKey`, `IsPFKey`, `XChangeKeyboardMapping`, `XDeleteModifiermapEntry`, `XDisplayKeycodes`, `XFreeModifiermap`, `XGetKeyboardMapping`, `XGetModifierMapping`, `XInsertModifiermapEntry`, `XKeysymToKeycode`, `XKeysymToString`, `XLookupKeysym`, `XLookupString`, `XNewModifierMap`, `XQueryKeymap`, `XRebindKeySym`, `XRefreshKeyboardMapping`, `XSetModifierMapping`, `XStringToKeysym`.

XKeysymToKeycode

Name

XKeysymToKeycode — convert a keysym to the appropriate keycode.

Synopsis

```
KeyCode XKeysymToKeycode(display, keysym)
    Display *display;
    Keysym keysym;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
keysym	Specifies the keysym that is to be searched for.

Description

`XKeysymToKeycode` returns the keycode corresponding to the specified keysym in the current mapping. If the specified keysym is not defined for any keycode, `XKeysymToKeycode` returns zero.

Related Commands

`IsCursorKey`, `IsFunctionKey`, `IsKeypadKey`, `IsMiscFunctionKey`, `IsModifierKey`, `IsPFKey`, `XChangeKeyboardMapping`, `XDeleteModifiermapEntry`, `XDisplayKeycodes`, `XFreeModifiermap`, `XGetKeyboardMapping`, `XGetModifierMapping`, `XInsertModifiermapEntry`, `XKeycodeToKeysym`, `XKeysymToString`, `XLookupKeysym`, `XLookupString`, `XNewModifierMap`, `XQueryKeymap`, `XRebindKeySym`, `XRefreshKeyboardMapping`, `XSetModifierMapping`, `XStringToKeysym`.

XKeysymToString

Name

XKeysymToString — convert a keysym symbol to a string.

Synopsis

```
char *XKeysymToString(keysym)
    KeySym keysym;
```

Arguments

keysym	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 "Shift".

Note that `XKeysymString` does not return the string that is mapped to the keysym, but only a string version of the keysym itself. In other words, even if the F1 key is mapped to the string "-STOP" using `XRebindKeysym`, `XKeysymToString` still returns "F1". `XLookupString`, however, would return "STOP".

In Release 4, `XKeysymToString` can process keysyms that are not defined by the Xlib standard. Note that the set of keysyms that are available in this manner and the mechanisms by which Xlib obtains them is implementation dependent. (In the MIT sample implementation, the resource file */usr/lib/X11/XKeysymDB* is used starting in Release 4. The keysym name is used as the resource name, and the resource value is the keysym value in uppercase hexadecimal.)

Related Commands

`IsCursorKey`, `IsFunctionKey`, `IsKeypadKey`, `IsMiscFunctionKey`, `IsModifierKey`, `IsPFKey`, `XChangeKeyboardMapping`, `XDeleteModifiermapEntry`, `XFreeModifiermap`, `XGetKeyboardMapping`, `XGetModifierMapping`, `XInsertModifiermapEntry`, `XKeycodeToKeysym`, `XKeysymToKeycode`, `XLookupKeysym`, `XLookupString`, `XNewModifierMap`, `XQueryKeymap`, `XRebindKeysym`, `XRefreshKeyboardMapping`, `XSetModifierMapping`, `XStringToKeysym`.

XKillClient

Name

XKillClient — destroy a client or its remaining resources.

Synopsis

```
XKillClient(display, resource)
    Display *display;
    XID resource;
```

Arguments

display	Specifies a connection to an X server; 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 `Retain-Temporary` mode, all of the client's resources are destroyed. If `AllTemporary` is specified in the *resource* argument, then the resources of all clients that have terminated in `Retain-Temporary` are destroyed.

For more information, see Volume One, Chapter 13, *Other Programming Techniques*.

Errors

`BadValue`

Related Commands

`XSetCloseDownMode`.

XListDepths

Name

XListDepths — determine the depths available on a given screen.

Synopsis

```
int *XListDepths(display, screen_number, count)
      Display *display;
      int screen_number;
      int *count;      /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
screen_number	Specifies the appropriate screen number on the host server.
count	Returns the number of depths.

Availability

Release 4 and later.

Description

`XListDepths` returns the array of depths that are available on the specified screen. If the specified *screen_number* is valid and sufficient memory for the array can be allocated, `XListDepths` sets *count* to the number of available depths. Otherwise, it does not set *count* and returns `NULL`. To release the memory allocated for the array of depths, use `XFree`.

Related Commands

`DefaultDepthOfScreen` macro, `DefaultDepth` macro, `XListPixmapFormats`.

XListExtensions

Name

XListExtensions — return a list of all extensions to X supported by Xlib and the server.

Synopsis

```
char **XListExtensions(display, nextensions)
    Display *display;
    int *nextensions;          /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
nextensions	Returns the number of extensions in the returned list.

Description

`XListExtensions` lists all the X extensions supported by Xlib and the current server. The returned strings will be in ISO LATIN-1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.

For more information on extensions, see Volume One, Chapter 13, *Other Programming Techniques*.

Related Commands

`XFreeExtensionList`, `XQueryExtension`.

XListFonts

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 connection to an X server; returned from `XOpenDisplay`.
pattern	Specifies the string associated with the font names you want returned. You can specify any string, including asterisks (*), and question marks. The asterisk indicates a wildcard for any number of characters and the question mark indicates a wildcard for a single character. Upper or lower case is not important. The string should be in ISO LATIN-1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.
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 returned font name string is terminated by `NULL` and is lower case. The maximum number of names returned in the list is the value you passed to *maxnames*. The function returns the actual number of font names in *actual_count*.

If no fonts match the specified names, `XListFonts` returns `NULL`.

The client should call `XFreeFontNames` when done with the font name list.

The font search path (the order in which font names in various directories are compared to *pattern*) is set by `XSetFontPath`.

For more information on fonts, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Related Commands

`XCreateFontCursor`, `XFreeFont`, `XFreeFontInfo`, `XFreeFontNames`, `XFreeFontPath`, `XGetFontPath`, `XGetFontProperty`, `XListFontsWithInfo`, `XLoadFont`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XListFontsWithInfo

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 connection to an X server; returned from `XOpenDisplay`.
pattern	Specifies the string associated with the font names you want returned. You can specify any string, including asterisks (*) and question marks. The asterisk indicates a wildcard on any number of characters and the question mark indicates a wildcard on a single character. Upper or lower case is not important. The string should be in ISO LATIN-1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.
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 a pointer to a list of font information structures. `XListFontsWithInfo` provides enough space for *maxnames* pointers.

Description

`XListFontsWithInfo` returns a list of font names that match the specified *pattern* and a also returns limited information about each font that matches. The list of names is limited to the size specified by the *maxnames* argument. The list of names is in lower case.

`XListFontsWithInfo` returns `NULL` if no matches were found.

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.

The font search path (the order in which font names in various directories are compared to *pattern*) is set by `XSetFontPath`. `XListFonts` returns `NULL` if no matches were found.

For more information on fonts, see Volume One, Chapter 6, *Drawing Graphics and Text*.

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 nonzero size*/
     unsigned default_char;        /* char to print for undefined character */
     int n_properties;             /* how many properties there are */
     XFontProp *properties;        /* pointer to array of additional properties*/
     XCharStruct min_bounds;       /* minimum bounds over all existing char*/
     XCharStruct max_bounds;       /* 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;
```

Related Commands

XCreateFontCursor, XFreeFont, XFreeFontInfo, XFreeFontNames, XFree-FontPath, XGetFontPath, XGetFontProperty, XListFonts, XLoadFont, XLoadQueryFont, XQueryFont, XSetFont, XSetFontPath, XUnloadFont.

XListHosts

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 connection to an X server; returned from `XOpenDisplay`.
nhosts	Returns the number of hosts currently in the access control list.
state	Returns whether the access control list is currently being used by the server to process new connection requests from clients. `True` if enabled, `False` if disabled.

Description

`XListHosts` returns the current access control list as well as whether the use of the list is enabled or disabled. `XListHosts` allows a program to find out what machines make connections, by looking at a list of host structures. This `XHostAddress` list should be freed when it is no longer needed. `XListHosts` returns `NULL` on failure.

For more information on access control lists, see Volume One, Chapter 13, *Other Programming Techniques*.

Structures

```
typedef struct {
    int family;
    int length;
    char *address;
} XHostAddress;
```

Related Commands

`XAddHost`, `XAddHosts`, `XDisableAccessControl`, `XEnableAccessControl`, `XRemoveHost`, `XRemoveHosts`, `XSetAccessControl`.

XListInstalledColormaps

Name

XListInstalledColormaps — get a list of installed colormaps.

Synopsis

```
Colormap *XListInstalledColormaps(display, w, num)
      display *display;
      Window w;
      int *num;                    /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
w	Specifies the ID of 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 containing 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.

`XListInstalledColormaps` returns `None` and sets *num* to zero on failure.

The allocated list should be freed using `XFree` when it is no longer needed.

For more information on installing colormaps, see Volume One, Chapter 7, *Color*.

Errors

`BadWindow`

Related Commands

`DefaultColormap`, `DisplayCells`, `XCopyColormapAndFree`, `XCreateColormap`, `XFreeColormap`, `XGetStandardColormap`, `XInstallColormap`, `XSetStandardColormap`, `XSetWindowColormap`, `XUninstallColormap`.

XListPixmapFormats

Name

XListPixmapFormats — obtain the supported pixmap formats for a given server.

Synopsis

```
XPixmapFormatValues *XListPixmapFormats(display, count)
      Display *display;
      int *count;      /* RETURN */
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
count	Returns the number of pixmap formats that are supported by the server.

Availability

Release 4 and later.

Description

`XListPixmapFormats` returns an array of `XPixmapFormatValues` structures that describe the types of Z format images that are supported by the specified server. If insufficient memory is available, `XListPixmapFormats` returns `NULL`. To free the allocated storage for the `XPixmapFormatValues` structures, use `XFree`.

Structures

```
typedef struct {
    int depth;
    int bits_per_pixel;
    int scanline_pad;
} XPixmapFormatValues;
```

Related Commands

`XListDepths`.

XListProperties

Name

XListProperties — get the property list for a window.

Synopsis

```
Atom *XListProperties(display, w, num_prop)
      Display *display;
      Window w;
      int *num_prop;                  /* RETURN */
```

Arguments

display	Specifies a connection to an X server; 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 atoms for properties that are defined for the specified window. `XListProperties` returns `NULL` on failure (when window *w* is invalid).

To free the memory allocated by this function, use `XFree`.

For more information, see Volume One, Chapter 10, *Interclient Communication*.

Errors

`BadWindow`

Related Commands

`XChangeProperty`, `XDeleteProperty`, `XGetAtomName`, `XGetFontProperty`, `XGetWindowProperty`, `XInternAtom`, `XRotateWindowProperties`, `XSet-StandardProperties`.

XLoadFont

Name

XLoadFont — load a font if not already loaded; get font ID.

Synopsis

```
Font XLoadFont(display, name)
      Display *display;
      char *name;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
name	Specifies the name of the font in a null terminated string. As of Release 4, the * and ? wildcards are allowed and may be supported by the server. Upper or lower case is not important. The string should be in ISO LATIN-1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.

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, 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 font 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.

For more information on fonts, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

`BadAlloc`	Server has insufficient memory to store font.
`BadName`	*name* specifies an unavailable font.

Related Commands

`XCreateFontCursor`, `XFreeFont`, `XFreeFontInfo`, `XFreeFontNames`, `XFreeFontPath`, `XGetFontPath`, `XGetFontProperty`, `XListFonts`, `XListFontsWithInfo`, `XLoadQueryFont`, `XQueryFont`, `XSetFont`, `XSetFontPath`, `XUnloadFont`.

XLoadQueryFont

Name

XLoadQueryFont — load a font and fill information structure.

Synopsis

```
XFontStruct *XLoadQueryFont(display, name)
    Display *display;
    char *name;
```

Arguments

display	Specifies a connection to an X server; returned from `XOpenDisplay`.
name	Specifies the name of the font. This name is a null terminated string. As of Release 4, the * and ? wildcards are allowed and may be supported by the server. Upper or lower case is not important.

Description

`XLoadQueryFont` performs an `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.

For more information on fonts, see Volume One, Chapter 6, *Drawing Graphics and Text*.

Errors

`BadAlloc`	server has insufficient memory to store font.
`BadName`	`name` specifies an unavailable 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 nonzero size*/
    unsigned default_char;       /* char to print for undefined character */
    int n_properties;            /* how many properties there are */
    XFontProp *properties;       /* pointer to array of additional properties*/
    XCharStruct min_bounds;      /* minimum bounds over all existing char*/
    XCharStruct max_bounds;      /* 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 character */
    short rbearing;              /* origin to right edge of character */
    short width;                 /* advance to next char's origin */
    short ascent;                /* baseline to top edge of character */
    short descent;               /* baseline to bottom edge of character */
    unsigned short attributes;   /* per char flags (not predefined) */
} XCharStruct;
```

Related Commands

XCreateFontCursor, XFreeFont, XFreeFontInfo, XFreeFontNames, XFree-FontPath, XGetFontPath, XGetFontProperty, XListFonts, XListFontsWith-Info, XLoadFont, XQueryFont, XSetFont, XSetFontPath, XUnloadFont.

XLookUpAssoc

Name

XLookUpAssoc — obtain data from an association table.

Synopsis

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

Arguments

display	Specifies a connection to an X server; 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 locally and accessing by ID. `XLookUp-Assoc` 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`.

For more information on association tables, see Volume One, Appendix B, *X10 Compatibility*.

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;
```

Related Commands

`XCreateAssocTable`, `XDeleteAssoc`, `XDestroyAssocTable`, `XMakeAssoc`.

XLookupColor

Name

XLookupColor — get database RGB values 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 connection to an X server; returned from `XOpenDisplay`.
cmap	Specifies the colormap.
colorname	Specifies a color name string (for example "red"). Upper or lower case does not matter. The string should be in ISO LATIN1 encoding, which means that the first 128 character codes are ASCII, and the second 128 character codes are for special characters needed in western languages other than English.
rgb_db_def	Returns the exact RGB values for the specified color name from the */usr/lib/X11/rgb* database.
hardware_def	Returns the closest RGB values possible on the hardware.

Description

`XLookupColor` looks up RGB values for a color given the colorname string. It returns both the exact color values and the closest values possible on tthe screen specified by *cmap*.

`XLookupColor` returns nonzero if *colorname* exists in the RGB database or zero if it does not exist.

To determine the exact RGB values, `XLookupColor` uses a database on the X server. On UNIX, this database is */usr/lib/X11/rgb*. To read the colors provided by the database on a UNIX-based system, see */usr/lib/X11/rgb.txt*. The location, name, and contents of this file are server-dependent.

For more information see Volume One, Chapter 7, *Color*, and Appendix D, *The Color Database*, in this volume.

Errors

`BadName`	Color name not in database.
`BadColormap`	Specified colormap invalid.

Structures

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

Related Commands

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

XLookupKeysym

Name

XLookupKeysym — get the keysym corresponding to a 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 symbols `ShiftMap-Index`, `LockMapIndex`, `ControlMapIndex`, `Mod1MapIndex`, `Mod2-MapIndex`, `Mod3MapIndex`, `Mod4MapIndex`, and `Mod5MapIndex` can be used.

Description

Given a keyboard event and the *index* into the list of keysyms for that keycode, `XLookup-Keysym` returns the keysym from the list that corresponds to the keycode in the event. If no keysym is defined for the keycode of the event, `XLookupKeysym` returns `NoSymbol`.

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.

Structures

```
typedef struct {
    int type;               /* of event */
    unsigned long serial;   /* # of last request processed by server */
    Bool send_event;       /* true if this came from a SendEvent request */
    Display *display;       /* display the event was read from */
    Window window;          /* "event" window it is reported relative to */
    Window root;            /* root window that the event 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;
```

Related Commands

XChangeKeyboardMapping, XDeleteModifiermapEntry, XFreeModifiermap, XGetKeyboardMapping, XGetModifierMapping, XInsertModifiermapEntry, XKeycodeToKeysym, XKeysymToKeycode, XKeysymToString, XLookupString, XNewModifierMap, XQueryKeymap, XRebindKeysym, XRefreshKeyboard-Mapping, XSetModifierMapping, XStringToKeysym.

XLookupString

Name

XLookupString — map a 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 X Consortium Xlib through Release 4.

Description

`XLookupString` gets an ASCII string and a keysym that are currently 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 *buffer*. The length may be greater than 1 if the event's keycode translates into a keysym that was rebound with `XRebindKeysym`.

The compose *status* is not implemented in any release of the X Consortium version of Xlib through Release 4.

In Release 4, `XLookupString` implements the new concept of keyboard groups. Keyboard groups support having two complete sets of keysyms for a keyboard. Which set will be used can be toggled using a particular key. This is implemented by using the first two keysyms in the list for a key as one set, and the next two keysyms as the second set. For more information on keyboard groups, see Volume One, Appendix G, *Release Notes*.

For more information on using `XLookupString` in general, see Volume One, Chapter 9, *The Keyboard and Pointer*.

Structures

```
/*
 * Compose sequence status structure, used in calling XLookupString.
 */
```